Performance Analysis of Transactional Traffic in Mobile Ad-hoc Networks

Yufei Cheng

Submitted to the graduate degree program in Electrical Engineering & Computer Science and the Graduate Faculty of the University of Kansas School of Engineering in partial fulfillment of the requirements for the degree of Master of Science

Thesis Committee:

Dr. James P.G. Sterbenz: Chairperson

Dr. Gary Minden

Dr. Fengjun Li

Date Proposed

The Thesis Committee for Yufei Cheng certifies
that this is the approved version of the following thesis:

Performance Analysis of Transactional Traffic in Mobile Ad-hoc Networks

Committee:

Dr. James P.G. Sterbenz: Chairperson

Dr. Gary Minden

Dr. Fengjun Li

Date Approved

Abstract

Mobile Ad Hoc networks (MANETs) present unique challenge to new protocol design, especially in scenarios where nodes are highly mobile. Routing protocols performance is essential to the performance of wireless networks especially in mobile ad-hoc scenarios. The development of new routing protocols requires comparing them against well-known protocols in various simulation environments. The protocols should be analysed under realistic conditions including, but not limited to, representative data transmission models, limited buffer space for data transmission, sensible simulation area and transmission range combination, and realistic moving patterns of the mobiles nodes. Furthermore, application traffic like transactional application traffic has not been investigated for domain-specific MANETs scenarios. Overall, there are not enough performance comparison work in the past literatures. This thesis presents extensive performance comparison among MANETs comparing transactional traffic including both highly-dynamic environment as well as low-mobility cases.

I like to dedicate this work to my parents for their continuous support and guidance with which I could reach this level.

Acknowledgements

I would like to thank my committee members especially my advisor Dr. James P.G. Sterbenz for his continuous support and illuminating instructions. I would also like to thank the PhD students Abdul Jabbar, Justin Rohrer and Egemen K. Çetinkaya in our group for mentoring and advising me in the process of the thesis as well as the research works, and Hemanth Narra for taking lead in DSDV implementation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

Over the past decades, there has been tremendous development in mobile devices, such as mobile phones, laptops, and automobiles or aircrafts carrying wireless transmission devices. Traditionally, there are wired infrastructures carrying information and forming network connecting the mobile nodes. Nowadays, there are increasing number of circumstances when there is no preestablished infrastructures, such as in emergency rescue or military operations. Networks that operate in this kind of environment are known as mobile ad hoc networks (MANETs) [1]. MANETs are a self-configuring network of mobile nodes connected by wireless links to form an arbitrary topology without the use of pre-existing infrastructure or centralized administration. It can be characterised by the dynamic topology, limited resources, and lack of fixed infrastructures. Nodes in MANETs change their position arbitrarily and they perform as both end systems and routers. All these characteristics present great challenges to the network protocol design, especially the routing and transport protocol design from which the challenges mainly come from two sources. The first one is the wireless channel with limited and varied bandwidth caused by interference and noise. Because of heterogeneous nodes

and different up- and down-link characteristics, the network connectivity is usually asymmetric. It invalidates some of the assumptions from wired network protocols in which most of the packet drops come from network congestion; instead, most of the packet losses are caused by corruption. The other challenge comes from the mobility of the nodes, which results dynamic topology with nodes frequently moving out of range from each other. As a result, the routing protocol design must cope with the dynamic topology, either reactively or proactively. The traditional MANETs routing protocols are designed to tolerate slowly moving nodes, mostly at lower than 20 m/s [2,3]. With the dynamic topology, the links between different pair of nodes break and most of the routing protocols use route caching to save the route entries, and entries expire faster when nodes are moving faster. A new round of route discovery needs to be initiated and this introduces more routing overhead. The higher the node speed, the more control packets need to be sent out and requiring extra network resource. Efficient routing protocols can provide significant benefits to MANET in terms of both performance and dependability.

Domain specific networks are gaining popularity recently, such as the aeronautical environment with multi-Mach velocity mobile nodes and satellite networks with high error-rate link and high round-trip delay. There are routing protocols designed and implemented specific for these domain specific networks. This work provides baseline comparison work for the design and comparison of new routing protocols. Besides different simulation environments with differing requirement on protocol design, different types of application traffic are expected to present different characteristics and pose different requirements for transport protocols as well as routing protocols. The canonical transport protocol to carry transactional traffic is HTTP on TCP, there are several drawbacks for TCP to better support

transactional traffic. Part of this thesis will analyse the relative performance of transactional traffic over different transport protocols and routing protocols.

1.1 Problem statement

Simulation has been the predominant evaluation methodology for MANET research [1, 4]. It provides easily accessible resources to analyse and evaluate new protocols and models. However, there has not been enough simulation comparison work in either the traditional MANET environment analysing different application traffic with different transport and routing protocol combinations, not to mention the aeronautical environment or satellite networks. In this thesis, we conduct a systematic comparison work for these scenarios. A large number of routing protocols for MANETs have been proposed [5–9], however, most the comparison works have been limited in terms of number of protocols, or biased in terms of limited choice of network scenarios to validate a particular proposed protocol. Some of the works only use one mobility model which can be easily biased due to the realistic of that specific mobility model [10].

A number of problems have been identified in the MANET simulation comparison works [4], the major ones include the repeatability, unbiasedness, rigorousness, and statistical soundness. There are many discrete-event network simulators available for the MANET community [11–14]. However, a considerable number of papers have not mentioned which simulation software they are using, and some do not mention the full set of parameter configurations. This poses great challenges for repeatability of their results and raises double for the rigorousness and unbiasedness of the simulation process and results. We plan to use one discrete event network simulator and provide as detailed parameters as possible to guarantee the

repeatability and carry out a rigorous performance comparison.

We use the ns-3 network simulator [15] to analyse protocol performance. The ns-3 simulator is a recent replacement for the open-source ns-2 simulator with many improvements, including modular extensibility and mixed wired and wireless models, arbitrary mix of link types and routing algorithms. However, the standard distribution lacks some well-known models [16]. The ns-2 simulator [11] has been widely used due to its large number of open-source models which is very useful for the academic research community. However, a major shortcoming of ns-2 is its limited scalability [17, 18] in terms of memory usage and simulation run-time, monolithic implementation, need to hack source, non-heterogeneous. In response to a number of ns-2 deficiencies, the ns-3 discrete event network simulator [15] has been developed, providing greater flexibility, modularity with C++, evolvability, and support for heterogeneity including hybrid wired and wireless models. The ns-3 discrete-event network simulator [15] is the successor of ns-2 and have a lot of advantages over ns-2.

Despite its advantages, ns-3 is relatively new with few protocol models yet incorporated into its release distribution [16]. Existing built-in traffic generators are limited to *BulkSendApplication* for bulk data transfer and *OnOffApplication* for constant bit rate (CBR) application, routing protocols are limited to AODV and OLSR. As part of our contribution from the ResiliNets group we have modeled DSDV routing protocol [19], 3D–Gauss-Markov mobility model [20], TDMA MAC protocol, DSR routing protocol [21], and HTTP traffic generator [22], TCP Westwood and Westwood+ [23]. This thesis demonstrates the performance comparison of different traffic types in MANET environment and how they interact with each other when using different routing protocols. For the traffic types, we

have used bulk data transfer, HTTP traffic and UDP traffic. We also carry out a combined traffic type which using TCP traffic as background traffic and measure how it affects the transmission of UDP traffic. As far as we know, this is the first work toward a comprehensive comparison of different traffic types in MANET environment.

1.2 Contributions

The contributions of this thesis are the following:

- Implement HTTP traffic generator in ns-3 to generate transactional traffic and test how different transport protocols and routing protocols perform.
- Implement DSR routing protocol to establish a canonical set of MANET routing protocols in ns-3 to compare AeroRP's performance.
- Assist Hemanth Narra with the implementation of DSDV in ns-3.
- Analyse the performance of AeroRP against DSDV, AODV, OLSR, and DSR in ns-3.
- Analyse the performance of different application traffic types including CBR, transactional traffic, and bulk data transfer in MANETs.
- Analyse how different mobility models and velocity affect the network performance.

1.3 Publications

This section highlights my publications over the course of my Masters program.

- Yufei Cheng, Egemen K. Çetinkaya, and James P.G. Sterbenz,"Transactional Traffic Generator Implementation in ns-3", in the *ICST SIMUTools Workshop on ns-3 (WNS3)* March 2013, Cannes, France, pp. 182–189.

- Yufei Cheng, Egemen K. Çetinkaya, and James P.G. Sterbenz, "Dynamic Source Routing (DSR) Protocol Implementation in ns-3", in the *ICST SIMUTools Workshop on ns-3 (WNS3)* March 2012, Sirmione, Italy.

- Hemanth Narra, Yufei Cheng, Egemen K. Çetinkaya, Justin P. Rohrer and James P.G. Sterbenz, "Destination-Sequenced Distance Vector (DSDV) Routing Protocol Implementation in ns-3", in the *4th International ICST Conference on Simulation Tools and Techniques, Wns3 2011* March 25, Barcelona, Spain.

- Yufei Cheng, Egemen K. Çetinkaya, and James P.G. Sterbenz, "Performance Comparison of Routing Protocols for Transactional Traffic over Aeronautical Networks", *International Telemetering Conference (ITC) 2011*, , October 2011, Las Vegas, NV

1.4 Organisation

The rest of the proposal is organised as follows. Chapter 2 presents background and related work, specifically the characteristics and challenges involved with the Web traffic simulation in MANETs. Implementation details of DSR routing protocol and HTTP traffic generator in ns-3 are explained in Chapter 3. Simulation results of analysing the performance of three types of application traffic on a variety of MANET routing protocols as well as transport protocols are discussed in

Chapter 4. Chapter 5 presents the conclusion of this work and the possible future work following this thesis.

Chapter 2

Background and Related Work

MANETs present unique characteristics and challenges to protocol design. We need to understand how different application types, transport protocols, routing protocols interact with each other and present performance analysis with different protocol combinations for a fair baseline comparison case to aid further protocol design. Furthermore, the trend of highly dynamic wireless nodes forming network presents new challenges to the protocol design. There is not enough work in comparing performance of different routing protocols when carrying different application types. Therefore, we present this work to provide a detailed performance comparison with different protocol combinations. We also provide model implementations including Dynamic Source Routing (DSR) protocol and transactional traffic generator implementations in ns-3 simulator, and verified their performances.

The application traffic types we compare are constant bit rate (CBR), bulk data transfer, and transactional traffic (HTTP). CBR traffic has been widely used to compare protocol performance [2, 7, 19], and we use it as the baseline performance comparison case. Bulk data transfer is mainly used to test transport

protocol performance [24–26], as it tests the time for the source to transfer a large chunk of data to the destination node. While transactional traffic would be used to analyse how transport protocol perform carrying real world web traffic. There are some past works about simulation with transactional traffic over different transport protocol as well as routing protocols [27, 28].

The routing protocols we intended to compare include AODV [5], DSDV [6], DSR [7], OLSR [8]. Routing protocol performance is commonly analysed with constant bit rate (CBR) traffic and bulk data transfer models; however, the network loads from Web-based applications is typically characterised by bursty traffic flowing over connections with variable durations ranging from a few seconds to several minutes. Web access consists of frequent, short request-response style traffic based on bursts of many small requests and frequently large responses. Web users tend to surf rapidly among sites, which are verified from both client [27] and server-side traces [29].

This chapter is organised as follows. Section 2.1 introduces HTTP protocol and its characteristics. Section 2.2 introduced the characteristics and challenges of simulation when transactional traffic involved and Section 2.3 briefly discusses the traffic generators developed in the network simulation community. Section 2.4 provides an introduction of several important routing protocols in MANETs.

2.1 Hypertext Transfer Protocol

HTTP (Hypertext Transfer Protocol) is a stateless request-response application layer protocol using the client-server paradigm for end-to-end data transfer. Assuming the transport protocol is TCP, It operates as the client initiates transport connections to the server by sending a SYN message; upon receiving the SYN

message, the server responds with a SYN ACK message to the client. When receiving the SYN ACK message, the client sends the HTTP request to the server, which also serves as the purpose of the ACK for the SYN acknowledgement message. When the server receives the request from the client, the TCP three way handshake is finished and started the transport connection. Furthermore, the server sends the web page the client is requesting. Here we would like to introduce pipelining and the persistent connections. The original version, HTTP 1.0 [30], uses a separate connection for each request-response transaction. HTTP 1.1 [31] uses *persistent* connections to reduce the latency by allowing several request-response messages over a single TCP connection. Another performance enhancement in HTTP 1.1 is the *pipelining* of a series of requests on a persistent connection without waiting for a response between requests, greatly improving HTTP performance. Another feature of HTTP 1.1 is that it can establish multiple parallel TCP connections between a sender-receiver pair to substantially increase TCP throughput. HTTP plays a key role in the communication of web browsers with web servers and ensures the secure communication channel.

2.2 Simulation Characteristics involved with HTTP Traffic

Transactional traffic generator is an essential part of network simulation, especially when the Internet traffic is considered. The generator is responsible for injecting synthetic network traffic into the simulation according to a model of how application users would behave in certain circumstances. As one major contributor of transactional traffic, HyperText Transfer Protocol (HTTP) is a pervasive application protocol and consumes a significant share of application flows in the Internet. HTTP traffic has transformed from mainly plain-text Web pages to large

size Web pages with a large number of embedded objects. With the sustaining influence of HTTP over the Web, the need for a HTTP source traffic model to accurately represent and simulate Web traffic has increased.

Simulating network behaviors is difficult because of its heterogeneity, sustained changes over time, and the immense variables attributed to the different types of applications in use. The heterogeneity of the Internet including different kinds of network traffic, different protocols dictating how one application should work, and the mixture of various applications web users prefer. Even worse is that all those heterogeneous parts of the Internet keep evolving in terms of both scale and diversity. The design methodology and assumption we use may be invalid in the near future. The author in [32] points out another major problem with network simulation is the traffic generation modeling, especially for a large simulation with realistic traffic mix. The author explains several invariables from the ever-changing Internet traffic, such as self-similarity, heavy-tailed distributions, and log-normal connection sizes. Thus, researchers are supposed to consider these essential invariables when simulating networks to represent the heterogeneity and scale-increasing nature of the Internet.

There are problems with statistical techniques in simulations involving Internet-like heavy-tailed workloads, which means the distribution where the tail follows a power law. It is one of the constants of the Internet. The approaches researchers have been using include Bounded Pareto [33], approximate Pareto with Lognormal [34], and treat data resulting from *heavy-tailed* workload as transit [35]. All three approaches have their relative advantages and disadvantages; improper use of them will decrease reproducibility of the simulation. For example, when treating data as transit, large sample of variables from the distribution function may

occur; thus, simulation time needs to be long enough to have reasonable results.

Traffic generators have been an essential part of simulations involving the Internet. Newer application types such as peer-to-peer applications start to consume a large share of network resources. However, considering the measured quantity of packets and bytes transmitted, web traffic is still a dominant application [36, 37]. HTTP traffic consumes a large share of application flows in the Internet, also it has transformed from a single web page transfer to large embedded object sharing, such as multimedia file sharing. One measurement study in April 2003 by Sprint shows that in 16 out of their 19 OC-48 links, web application consumes 59% of the total bandwidth [38]. Other recent studies have shown that HTTP traffic accounts for over 50% of the Internet traffic [36,36,37,39]. Kotz and Essien, as part of their measurements, have reported that 50% of wireless traffic on one university campus used HTTP's well know ports. [40, 41] Thus, when performing network experiments and simulations involving the Web usage, it is essential to consider the effect of web traffic on proposed mechanisms or protocols.

2.2.1 Web Traffic Characteristics

Web access is frequent, short request-response transactions based on bursts of many small Web requests and responses. Response messages are small to keep transmission time down; embedded objects per Web page and the frequency of smaller objects are increasing [32]. Although there are some very large responses, 85% of all responses are 10,000 bytes or less [42]; over 90% of the requests are between 100 and 1,000 bytes in size. There is also a year-to-year increase in the number of embedded objects on each Web page [32]. To make the burstiness worse, Web users tend to switch rapidly from site to site, as can be verified from both

client [27] and server side traces [29]. In addition, there are other user interactions such as clicking the browser *stop* or *reload* buttons while one page is loading [32]. All these characteristics contribute to the bursty nature of Web traffic.

Several works have shown that Web traffic is statistically self-similar [43–45], that is bursty on several or all time scales. To be able to accurately represent Internet traffic, we need to generate traffic that carries self-similarity features. We choose the source variable generation model from PackMime-HTTP [38], which has been verified to be able to generate self-similar traffic that matches real trace data.

HTTP is a stateless application protocol using the client-server paradigm. Its operation is transactional as the client sends one or a series of requests to the server, and the server responds to the request with response messages. HTTP is an application layer protocol and presumes a reliable TCP transport layer for end-to-end data transfer. The original version HTTP 1.0 [30] uses a separate connection for each request-response transaction. HTTP 1.1 [31] introduces three major performance enhancement mechanisms, with the first one using *persistent connections* to reduce the latency by allowing several request-response messages over a single TCP connection. The second performance enhancement in HTTP 1.1 is the *pipelining* of a series of requests on a persistent connection without waiting for responses for the previous requests. Note that the *pipelining* option can only be enabled when the *persistent connection* option is used. Thirdly, the *parallel connection* option is another important enhancement. It allows multiple transport connections opened for one Web page transaction and further lowers the loading latency. Our model does not support this later option in this release. We plan to incorporate it in the next release and compare its performance with

that of other options.

There are mainly three types of transactional traffic generators: *page-based*, *behavior-based*, and *connection-based*. Page-based traffic generators only focus on the Web page details and fail to represent other major characteristics of HTTP traffic [32], such as the server delay time and Web request gap time. On the other hand, behavior-based generators [46] simulate the ON/OFF states in which the ON state represents active Web-request and OFF state represents the silent period after all objects are retrieved. However, these models fail to accurately represent the transport connection characteristics, such as the gap time among opening new transport connections.

Connection-based models [38] provide a better alternative and their advantages have been shown [35, 47]. A connection-based implementation models TCP connections in terms of connection establishment rates, request/response data sizes, and gap time among objects within the connection. Synthetic traffic is injected into the simulation according to the distribution models of how users would behave in Web browsing activities. We developed our HTTP traffic model based on TCP connections between Web servers and clients, with each node acting as either server, client, or both. The model can run over both wired and wireless networks simulated with node mobility.

Scalability is another major factor for HTTP traffic generation to cope with ever-changing Web traffic characteristics. The *page-based* model [48] successfully captures Web traffic behavior when the model was implemented; however, it focuses on user-browsing behavior and constructs detailed Web pages. As Web traffic continues to change, the model fails to represent new Web browsing behavior as well as increasing size of embedded objects in Web pages. Therefore, a

traffic model with scalability is necessary to capture Web-browsing behavior, as well as to predict its behavior in the future. Furthermore, peer-to-peer traffic is gradually taking a leading role in network traffic. The *connection-based* model [38] is scalable to Web traffic evolvement and and can incorporate a peer-to-peer traffic model when needed.

2.3 Transactional Traffic Generators

Web traffic shows bursty characteristics. The duration of a flow ranges from a few seconds to a few minutes while the packet size distribution consists of a high proportion of 1500 B for data packets and 40 B for ACK packets [38]; however, large responses can reach to more than 100,000 B [32]. In our HTTP traffic generator implementation, we consider this wide range of traffic characteristics by tuning various simulation parameters. We model TCP connections in terms of connection rate, the timing of request and response messages, as well as the size of file transferred, and generate both HTTP 1.0 and HTTP 1.1 traffic. Our HTTP model can either generate real-world HTTP traffic based on pre-defined distributions or synthetic distributions based on user-specified values.

There are two major problems in previous works when modeling HTTP traffic generator. First, data from the two pioneering measurement projects capturing web-browsing behaviors, the Mah [49], and Barford et al. [50]studies, contributed to some of the web traffic generators currently in use. For both of the measurement studies, the population of users were quite distinctive but size of the traces gathered were relatively small. Another problem with all the traces is that they are relatively old, and these studies were conducted before the deployment of HTTP 1.1; therefore, their data captured can only represent HTTP 1.0 traffic

characteristics.

Another problem with some models is that they focus on HTTP page details too much and failed to represent major factor of HTTP traffic. A contemporary measurement study of web traffic produced model for page-based traffic generation by Smith, et al. [32]. In the case of a *page-based* model, in which one constructs a web page and simulates user behaviors when surfing the Internet is not sufficient for HTTP traffic generation in simulation. Forty percent of all data by web servers is from persistent connections. With a large amount of web traffic using persistent connections, the traffic model should involve persistent connection feature to represent current Internet traffic; however, the authors in [46] defined a detailed HTTP model with most of the parameters essential to the implementation of synthetic traffic generation. They assume that all the parameters in their model are independent of each other and traffic is simulated in an ON/OFF source, meaning it will divide the user browsing behaviors into ON and OFF time. However, these models focus on details of the web page itself, and important correlations between different variables are not considered.

There are broadly two types of transactional traffic generators: *page-based* and *connection-based*. *Page-based* traffic generators only focus on the HTTP page details and fail to represent other major characteristics of the HTTP traffic [32, 46]. Those models consider HTML page structures, HTTP status code, and some other details of HTTP protocol, which trivializes the core function of the HTTP model. Furthermore, modeling HTTP protocol based on page structures complicates the traffic generation model with details insignificant to overall performance. The ON/OFF model proposed in [51] has proved that in spite of ignoring traffic interactions through resource limitations and feedback control, it has been successful

in modeling observed Web traffic characteristics [52]. The model in [51] ignores interaction among traffic sources contending for network resources which can be as complicated as the feedback congestion control algorithm of TCP Reno in real networks. The file size distribution and traffic self-similarity is not significantly affected by changes in network resources, topology, traffic mixing, or the distribution of inter-arrival times. On the other hand, *connection-based* models provide a better alternative than page-based models [38] and their effectiveness has been proved in [35, 47].

Connection-based transactional traffic generator [38] models TCP connections in terms of connection establishment rates, request-response data sizes, and timing within the connection. They claim that modeling TCP connections in terms of connection establishment rates, sizes and timing of exchanges of request and response data within the connection could better model HTTP traffic and verified with simulation results. We developed our HTTP traffic model based on TCP connections between web servers and clients, with each node acting as either server, client, or both. The model can run over either wired or wireless networks, or even hybrid ones. They also list distributions of the traffic model parameters. For example, they explain the best-fit distribution for file sizes, server delay time, and in-line objects inter-arrival time. Several works [35,47] have demonstrated the effectiveness of this *connection-based* model. Furthermore, it is easier to replicate their results with the same distribution they used to generate HTTP source variables. We can tune the parameters of distributions to generate web workload that better represents current Internet traffic. Based on the *connection-based* model theory in [38], the authors have implemented it in the ns-2 network simulator. The model is validated at packet-level by comparing the measured BELL packet

traffic with synthetic packet traffic generated from simulation. The parameters used are the packet rates, number of active connections and connection duration, the inter-arrival time of web pages and the web object sizes.

However, there is a problem with the model in [38] as they put clients in one side of the network while servers on the other one. It also structures the model as if only one client is communicating with one server. This is a useful simplification in the case of a wired network since it can easily replicate the case for a campus local network connecting to the Internet. However, when comes to the case of wireless network, especially after introducing mobility to the hosts, the simplification in this model is not useful.

Therefore, we use the *connection-based* model to generate HTTP traffic based on TCP connections and develop our HTTP traffic model with individual web servers and clients into consideration. One node can act as either server or client based on the start-time configuration of simulation. This is advantageous as we can simulate both wired network and wireless network, with or without node mobility. For example, we can install HTTP traffic generator in some of the nodes in the network, and running other traffic types in the other nodes, which gives the traffic generator more flexibility in simulation process.

There are two working modes for our HTTP traffic generator, real-world traffic and manual mode. The difference between the two modes is how the source variables are generated. In the real-world mode, we generate each source variable automatically based on its relative stochastic distribution. Furthermore, all the distributions are calculated to represent two packet traces [32, 53]. We follow the source variables from the work [38]. This way we are able to generate web traffic that shares similar characteristics to the real-world traffic trace [32,53]. Therefore,

the network traffic we generated would be representing the real-world Internet traffic. However, for the users that care little about the real-world traffic, while want to generated user-specified traffic, there is another working mode. The other mode of traffic generation is user-defined mode, users can change all the source variable in command line. The model is designed to run simulations with detailed scenario settings from the users. For example, if we want to test how different web object sizes affect the network, we can tune the web object sizes while fixing all the other parameters. The two working modes are designed to suit most of the traffic generation operations.

Scalability to other types of traffic models is another major factor for HTTP traffic generation to cope with ever-changing web traffic characteristics. The *page-based* model in [46] successfully captures web traffic behavior when the model was implemented; however, it focuses on user-browsing behavior currently and constructs detailed web pages. With the changing of web traffic, the model fails to represent new web browsing behavior as well as ever-increasing size of embedded objects in web pages. Therefore, a traffic model with scalability is necessary to capture web-browsing behavior, as well as to predict the behavior in the future. Furthermore, peer-to-peer traffic is taking a leading role in network traffic since people are trying to upload traffic as well as download it. The model in [38] is scalable to web traffic evolvement and we can tune it to be a peer-to-peer traffic model when needed. Their view of network simulation follows the philosophy of using source-level descriptions of network traffic, in which one simulates the behavior of application users with transport connections. The traffic generator injects synthetic traffic into the simulation according to a distribution model of how users would behave in web browsing behavior for HTTP traffic generator. The gener-

ator can evolve to represent peer-to-peer application or any other transactional traffic type by changing the distribution model.

2.4 MANET Routing Protocols

The infrastructureless and the dynamic nature of MANETs poses a major challenge to accurate and efficient packet routing. This has led to a tremendous amount of research in routing protocols adaptable to the dynamic network conditions. Intelligent routing uses limited resources while at the same time adaptable to the changing network conditions such as: network size, traffic density, and network partitioning. Routing in MANETs is nontrivial as it possesses a couple of characteristics which makes them different from wired networks like the probability of errors is high due to impairments in transmission channel. The transmission power is normally low to conserve energy. Another challenge is mobility of nodes which causes frequent link breakage.

There are a great number of routing protocols proposed in the MANETs research community, and they can normally be classified into two categories, which include topology-base [54] and position-based routing protocols [55].

2.4.1 Topology-Based Routing Protocols

The topology-based routing protocols operate by identifying node neighbors or existing link-state information. There are a lot of them proposed and can be classified as proactive and reactive routing protocols based on the route discovery mechanism. The following sub-sections elaborate more on proactive, reactive routing protocols by taking examples from the prominent topology-based protocols.

2.4.1.1 Proactive Routing Protocols

Proactive routing protocols maintain routing information to all the nodes in the network. They add new routes or update existing ones by periodically distributing routing tables or by exchanging link-state information with each other. One advantage of it is that routes to any destination are immediately available when data transmission begins, which minimizes the packet delay. However, the tradeoff is the large overhead involved with flooding routing tables throughout the network. Two of the canonical proactive routing protocols for MANETs are DSDV and OLSR.

Destination-sequenced distance vector (DSDV) routing protocol [6] is a table-driven proactive routing protocol. It maintains a routing table with entries for all the nodes in the network, and provides a single path to the destination, which is selected using the distance vector shortest path routing algorithm– Bellman-Ford algorithm. The cost metric used is the *hop count*, which is the number of hops for the packet to reach its destination. In order to reduce the amount of overhead transmitted through the network, two types of update packets are used. These are referred to as a *full dump* and *incremental* packets. The full dump packet carries all the available routing information and the incremental packet carries only the information changed since the previous *full dump*. The *incremental* update messages are sent more frequently than the *full dump* packets. Due to these updates, the chances of having routing loops within the network increases. To eliminate routing loops, each update from the node is tagged with an independently chosen *sequence number*, and it must be incremented each time a periodic update is made by a single node. The sequence number of normal update must be an even number, since each time a periodic update is made, the

node increments its sequence number by 2 and adds its update to the routing message it transmits. Only if a node wants to send an update for an expired route to its neighbors, it increments the sequence number of the disconnected node by one. Nodes receiving this update with odd *sequence number* will remove the corresponding entry from the routing table. DSDV uses *settling time* to dampen the route fluctuations caused by mobility of the nodes.

Optimised link state routing (OLSR) protocol [8] is a point-to-point routing protocol based on the traditional link-state algorithm. OLSR uses HELLO and topology control (TC) messages to discover and broadcast *link state* information throughout the network regularly. Nodes receiving this topology information compute next hop destinations for all the nodes in the network. HELLO messages at each node discover 2-hop neighbor information and select a set of multi-point relays (MPRs). MPRs are responsible for transmitting broadcast messages and constructing link state information. Any node which is not in the set can process and read each packet but will not be able to retransmit it. Each node determines an optimal route (in terms of hops) to every known destinations using its topology information (from the topology table and neighboring table), and stores this information in a routing table. OLSR floods topology data frequently enough over the network to make sure all nodes are synchronised with *link state* information.

2.4.1.2 Reactive Routing Protocols

Reactive routing protocols discover routes only when needed and do not maintain routes to all the nodes in the network. They initiate a route request message to discover new routes when required. Nodes with routes to the destination will send back *route reply* message with the route information, which is later used to

send data packets. Route discovery is usually realized by flooding route request packets throughout the network. The main drawback of these protocols is the delay in discovering routes to new destinations. The problem may become worse in networks with highly dynamic nodes where nodes keep moving out of each other. AODV and DSR protocol are some of the well-known reactive routing protocols.

Ad hoc on-demand distance vector (AODV) [5,56] is a distance vector routing protocol that finds routes when needed. When a route does not exist to a given destination, a route request (RREQ) message is flooded by the source and by the intermediate nodes if they have no previous routes in their routing table. Upon receiving a RREQ message, the receiving node will record the route information in its own routing table. Once the RREQ message reaches the destination, the destination node responds by unicasting a route reply (RREP) message back to the neighbor from which it first received the RREQ message. As the RREP message is forwarded back along the reverse path, nodes along this path set up forwarding entries in their routing tables, pointing to the node from which they received RREP message. AODV uses *sequence numbers* created by the destination for each route entry to avoid routing loops. Routes with the largest sequence number are preferred when selecting routes from the source to the destination. The advantage of AODV is its adaptabilility to relatively highly dynamic networks. However, node may experience large delays during route construction. Link failure may initiate another route discovery, which introduces extra delays and consumes more bandwidth.

Dynamic source routing (DSR) [7,57] is an on-demand routing protocol similar to AODV. The key feature of it is employing the source-routing mechanism where each packet is required to carry all the node IP addresses from source to

destination. Route discovery and maintenance are the two major phases in DSR operation. It maintains a route cache containing source route entries to every other node in the network. When a source node wants to send a packet to a destination node, the source looks for a route to destination in its RouteCache. If a route is found, it attaches the source route to the packet header and forwards the packet along the route. The packet traverses all the nodes in the path specified by the source route all the way from source to destination. If a route is not found in the RouteCache, the source node initiates a route discovery process and broadcasts a RouteRequest message to every neighboring node (node within its transmission range). Nodes receiving the RouteRequest message check their RouteCache to see if they have a route to that destination. If a route is not found, they add their IP address to the RouteRequest header and re-broadcast the message. A source route record is formed in the header as the RouteRequest message is propagated throughout the network. When the RouteRequest reaches the destination, or when the intermediate node finds a route to the destination in its own RouteCache, it replies back with a RouteReply message containing the source route record copied from the RouteRequest message. The RouteReply message will traverse the path specified by the route record to reach the source node. On receiving the RouteReply message, the source node as well as all the intermediate nodes will update their route cache with this route record. The sender node adds source route information to the data packet and sends it out. Each node along the path is responsible for making sure the packet has reached the next hop node in the source route. If any intermediate node does not receive an acknowledgement from its next hop, the node should retransmit the packet. When the retransmission number reaches MaxMaintRexmt, a RouteError message is sent to the sender. Upon receiving the

route error message, all the nodes along the route will remove this broken link and use an alternate entry from its RouteCache. If necessary, it initiates a new route discovery process to the destination. An advantage of DSR is that nodes can store multiple routes in their route cache, which means that the source node can check its route cache for a valid route before initiating route discovery, and if a valid route is found there is no need for route discovery. This is very beneficial for a network with low mobility. Another one is that it does not require any periodical beaconing, saving bandwidth in the process.

2.4.2 Location-Based Routing Protocols

In a highly dynamic scenario, where network topology changes frequently and quickly, the control overhead involved with calculating routes can be extremely high, resulting in very low-performance networks. To prevent this from happening, a different forwarding paradigm more suitable for highly dynamic scenarios is needed. In contrast to topology-based routing methods, position-based forwarding protocols make routing decisions based on the geographical coordinates of nodes. Some example protocols are GPSR [58], LAR [59], SIFT [60], and DREAM [61]. Position-based routing protocols prove to be more efficient for highly dynamic scenarios. Our group is developing LAR and SIFT and we will include the comparison with location-based routing protocols in future works.

The algorithm for the location-based routing protocols identify nodes by their geo-location instead of IP addresses, and uses those coordinates to route greedily towards the destination. Position-based routing algorithm eliminates some of the limitations of topology-based routing protocols. A location service is used by the sender of a packet to determine the position of the destination node and to

include the position in the packet's destination address. The routing decision at each node is then based on the destination's position contained in the packet and the position of the forwarding node's neighbors. Therefore, position-based routing does not require the establishment or maintenance of the whole routes. The nodes do not have to either to store routing tables or to transmit messages to keep routing tables up to date. As a further advantage, position-based routing supports the delivery of packets to all nodes in a given geographic region naturally. This service is called geocasting.

Depending on the forwarding strategies, we can distinguish position-based routing into three main categories: greedy forwarding, restricted directional flooding, and hierarchical approaches. For the first two, a node forwards a given packet to one (greedy forwarding) or more (restricted directional flooding) one-hop neighbors that are located closer to the destination than the forwarding node itself. The third forwarding strategies is to form a hierarchy in order to scale to a large number of mobile nodes. We present the three forwarding strategy in the following.

2.4.2.1 ResTP Protocol Suite-AeroRP

The assumptions for which traditional MANET routing protocols are designed are not meeting the challenges posed by the aeronautical environment [62, 63]. Existing routing mechanisms either generate significant overhead or take long time to converge which is not suitable for highly dynamic telemetry networks. To cope with the challenges that the highly-dynamic environment, the AeroRP routing protocol [64, 65] takes advantage of geolocation information of airborne nodes (ANs) [65] to construct neighbor table for routing packets. Contrary to the other traditional MANET routing protocols that discover end-to-end paths,

AeroRP makes only per-hop routing decisions. This is very useful when the nodes in the airborne network move at very high velocities and often cause link breakage even before one end-to-end path is able to be established. The performance of the AeroRP protocol is further analysed and proved to outperform traditional MANET routing protocols in airborne telemetry network [64, 66].

AeroRP is essentially a proactive routing protocol yet takes extra mechanisms to limit the updates and therefore controls the routing overhead. It is able to exploit the cross-layer information from AeroNP, geolocation and topology information. It forwards data based on per-hop decisions and therefore avoids the necessity for global convergence, which makes it suitable for high-dynamic environments. However, there is a security limitation on the extent of using location information in telemetry network. Therefore, there are a couple of alternatives proposed with different extent of location information used in AeroRP routing protocol. It also offers different operating modes to balance among policies, security concern and the geolocation information needed for routing.

There are two major operations in AeroRP routing process. The first one is maintaining a list of available neighbors at any given point. The primary mechanism used is snooping to notify the presence of nodes in the network. The second part of the routing protocol is to find the best next hop neighbor to forward the data packet. There are three kinds of nodes in AeroRP operations, the ground stations (GS), the airborne nodes (ANs), and the relay nodes (RNs). The GSs are special nodes that are global sinks. They have the location information of every node either from the mission plan or constant tracking the ANs. RNs are the default next hop. It is efficient for the GS to follow only the relay node since it has very narrow beam width.

AeroRP employs some level of Quality of Service (QoS) and ensures the priority of command and control packets since the wireless links have limited capacities. It distinguishes data type from control type and gives high priority to control packets. It has also employed the congestion control mechanism. Nodes use the CI (congestion indicator) field to indicate its own congestion level. All packet transmissions carry the CI field along with the type and priority of the data. Neighboring nodes eavesdrop on the data transmission and are aware of the congestion level of the nodes. When a node is congested, neighboring node has data with less or the same priority will back off; data with higher priority gets through anyway. This mechanism successfully prevents the network from congestion.

2.4.3 Mobility Models

Due to the mobility characteristics of nodes, evaluation of the network performance needs a mobility model where node movements closely relates to or mimics the node movements in a real network [67] and numerous different mobility models have been proposed [68]. This poses great challenges in the analysis of MANETs protocols because the realistic motion scenarios are complex and vary from each other. For example, in a war zone, nodes might move in relation to a central node which might be control station co-ordinating the movement. Some soldiers might be on foot while others in vehicles introduce different node velocities [67]. On the other hand, in a conference meeting, the nodes would be expected to move only within a certain area and most probably at the same speed with the distances among each other small. Different mobility models are needed to evaluate the routing protocols in order to decide which one would be best for each scenario.

2.4.3.1 Random Walk

The Random Walk model was used to emulate the randomness of mobile nodes' movement. In Random Walk mobility model, the nodes can change direction after traveling for a specified amount of time or after traveling for a certain distance. They then choose a random direction and speed to repeat the process; if the node reaches the simulation boundary it bounces off at an angle that is determined by the incoming direction. The Random Walk model is one memoryless mobility as the current velocity and direction are not dependence on the previous ones. However, we observe that is not the case of mobile nodes in many real life applications, in this sense, it is not one realistic mobility model.

2.4.3.2 Random Waypoint

The Random WayPoint mobility model is probably the most commonly used mobility model in MANET simulation environments. A node chooses a random speed from a specified range, a random destination and a pause time. The node then moves towards the chosen destination with the specified speed and pauses for the specified amount of time before repeating the process until the end of the simulation. The speed and pause time for this mobility model can be defined one specific value or a range of values. Although its widely acceptance [2, 3], there is one dramatic drawback for this mobility model as its average speed is much lower than expected and the nodes tend to form in the middle of the simulation [10]. Steady-state Random WayPoint mobility model is a modification of Random WayPoint mobility model which was proposed to fix this issue [69]. In this work, we present simulation results for both of the mobility model.

2.4.3.3 Random Direction

The Random Direction mobility model is designed as an improvement for Random WayPoint mobility Model to overcome a density wave that is the clustering of all nodes in mostly the center of the simulation area. In this mobility model, the nodes choose a random direction and travel at a specified speed along this direction until it reaches the simulation boundary where it pauses for a specified amount of time before randomly choosing an angular direction between 0 and 180 degrees and repeating the process again. This ensures that a node has a very high probability area of traversing the whole simulation area. However, it has the drawback of being very unrealistic as the nodes are required to travel to the boundary of the simulation area and bounce back. It is also a memoryless mobility model

2.4.3.4 Gauss-Markov

The Gauss-Markov mobility model is a memory-based mobility model. Gauss Markov mobility model is a synthetic model and it combines random movements with memory unlike the other mobility models which tend to produce straight-line node movement due to lack of memory [20]. Every node begins at a random initial point and travels for a specified time interval known as a time step before changing both the speed and direction to newly calculated values. The new values are based on the previous ones. The 3-D version of this model in ns-3 was implemented by Dan Broyles of University of Kansas [20].

Chapter 3

Implementation of ns-3 Models

In this chapter, we present the implementation details of DSR routing protocol and HTTP traffic generation model. Section 3.1 presents a detailed explanation of DSR's headers, its route cache, transmitting and processing DSR advertisements, and data packet buffering. Section 3.3 details the implementation aspects of HTTP traffic generator. There are a set of different methodologies supporting the development of network Web traffic generators: *page-based*, *behavior-based*, and *connection-based* [38,43]. We chose the *connection-based* model for our traffic generator implementation. We extend and modify the source variable generation model from the Packmime-HTTP traffic generator [38] in ns-2 to work with ns-3. Furthermore, we add an extra working mode to our generator, which is the *user-defined* mode. In this working mode, users will be able to provide source variables to the HTTP transactions and be able to analyse the simulation scenarios they intend to test.

3.1 Implementation of DSR in ns-3

The MANET routing protocols in the initial release of ns-3 are limited to just the optimised link state routing (OLSR) and the ad hoc on-demand distance vector (AODV) protocols. Therefore, the ResiliNets group have developed an ns-3 implementation of the destination-sequenced distance vector (DSDV) [6, 19], as well as the dynamic source routing (DSR) protocol [7], and provide a baseline for performance comparisons for developing new protocols. The majority of this chapter has published in the WNS3 papers [21, 22]

This chapter is organised as follows. Section 3.1.1 presents the details of the DSR module implementation in ns-3. Implementation of DSR header is explained in Section 3.1.1.1. Section 3.1.1.2 talks about the working characteristics of DSR routing protocol, route discovery and route maintenance. Section 3.1.1.3 introduces the three types of acknowledgment mechanisms built in DSR. Section 3.1.1.5 illustrated the data structures used in this implementation.

3.1.1 DSR module for ns-3

This section describes our implementation of DSR. The two major components of the DSR operation are route discovery and route maintenance. All the major attributes used in this implementation are listed in Table 3.1. The relation among all the classes implemented in this module is shown in Figure 3.1.

We implemented the DSR routing protocol `ns3::dsr::DsrRouting` in ns-3 by extending from the abstract base class `ns3::Ipv4L4Protocol`. The `ns3::dsr::DsrFsHeader` and `ns3::dsr::DsrOptionHeader` is extended from `ns3::Header`, and they are essentially shim headers between transport layer and network layer. We have also declared `ns3::dsr::RouteCache` to store all the routes that have been dis-

covered in previous route discovery process. Similarly, we have declared the `ns3::dsr::SendBuffer` class to store all unsent data packets and `ns3::dsr::RreqTable` to avoid duplicate route requests as well as control the rate of consecutive route requests for one destination. The `ns3::dsr::MaintainBuff` is used to store the data packet when sent out from the send buffer and waiting for delivery of acknowledgment from the next hop node. An in-depth explanation of all these classes is presented in the following sections.

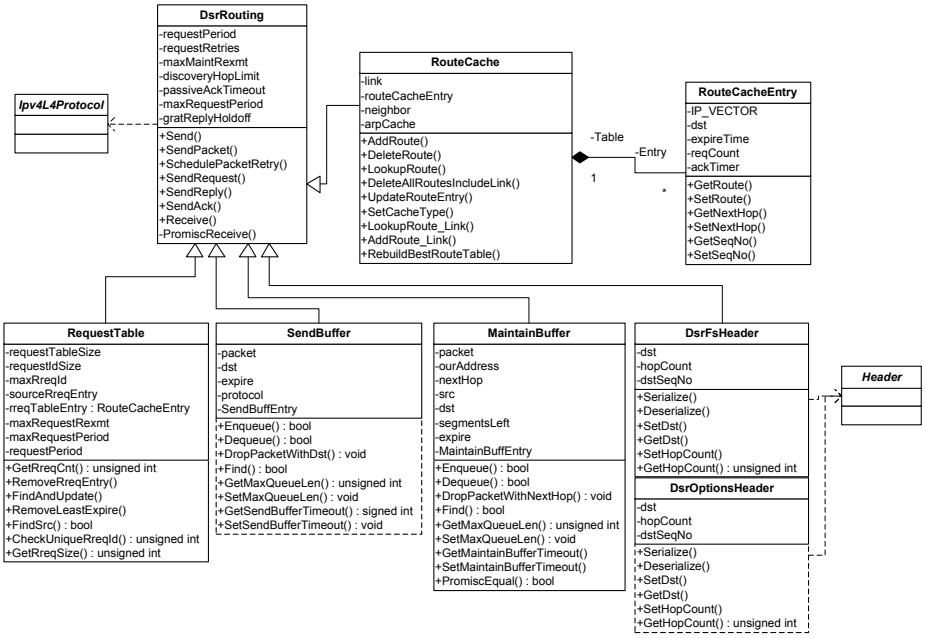

Figure 3.1. DSR class diagram

Most of the other routing protocol implementations in ns-3 are IP dependent, which renders plugging-in DSR as a shim between IP and the transport layer protocol problematic. In the current implementation of DSR in ns-3, it acts as `ns3::Ipv4L4Protocol` which uses the services of IP. Therefore, DSR should bypass IP's forwarding callback mechanism implemented in ns-3 and implement its own. To realize this, the destination address in IP header is always set as the

gateway address which is the next hop for the packet, and the real destination address will be shown in the DSR header. Figure 3.2 shows how DSR packets are encapsulated within IP in ns-3.

There are three fields added apart from default set up to deal with the tracing issue when analysing simulation results. Message id is used to identify the type of pa estination id are u f the packet, resp

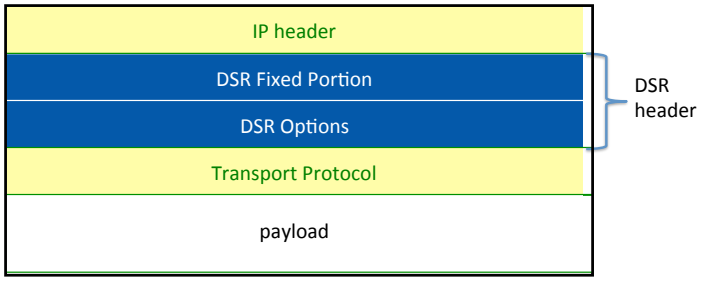

Figure 3.2. DSR header encapsulation within IP

3.1.1.1 DSR Header Format

The DSR header consists of two parts including `DsrFsHeader` and `DsrOptions Header`, as shown in Figure 3.3. The options shown after the fixed-header may be any one of the option headers. The `DsrFsHeader` is fixed in size and used to carry information that must be present in all of the DSR packets, while the `DsrOptionsHeader` is used to carry information for specific DSR options.

DSR Fixed-size Header The `DsrFsHeader` is a fixed size header which contains next header field to indicate the immediate header following the DSR option header. The payload length field indicates the length of all the option headers following the `DsrFsHeader`. The field message id, source id, and destination id are

added to resolve the ns-3 implementation issues. The modified header is shown in ler fie

Figure 3.3. Packet format for `DsrFsHeader`

The next header is a 4-bit field that indicates the upper layer protocol id, and indicates which transport protocol to pass the packet to. The message id field is also 4 bits in length and indicates the type of message this DSR header is carrying, message id number `1` means control packet, while id number `2` means data packet. This field is specifically created to resolve implementation issues in ns-3. The source id and destination id are also added only for ns-3 implementation to indicate the initiator and the destination of the packet since the source and destination field in IP header is changed as the packet transmitted hop-by-hop. Following is the payload length field which indicates the payload length of all the DSR option headers combined.

DSR Options Header The `DsrOptionsHeader` includes all the DSR options needed for protocol operation. The route request option, as shown in Figure 3.4, is attached to route discovery packet which will include all intermediate nodes' IP addresses in the header to form a full route to the destination. The route reply option is used to notify the source node with the whole route in its header with its header format shown in Figure 3.5.

The source route option is attached to data packets and is used to direct the packet from source to destination. It includes all intermediate nodes to form the full route. The header format is shown in Figure 3.6. When a route error occurs,

the route error option is used to indicate the link breakage and remove routes using that broken link from the route cache with its header format is shown in Figure 3.7.

The acknowledgement request option is used to request next hop network ac-

knc wl-

edg he

nex

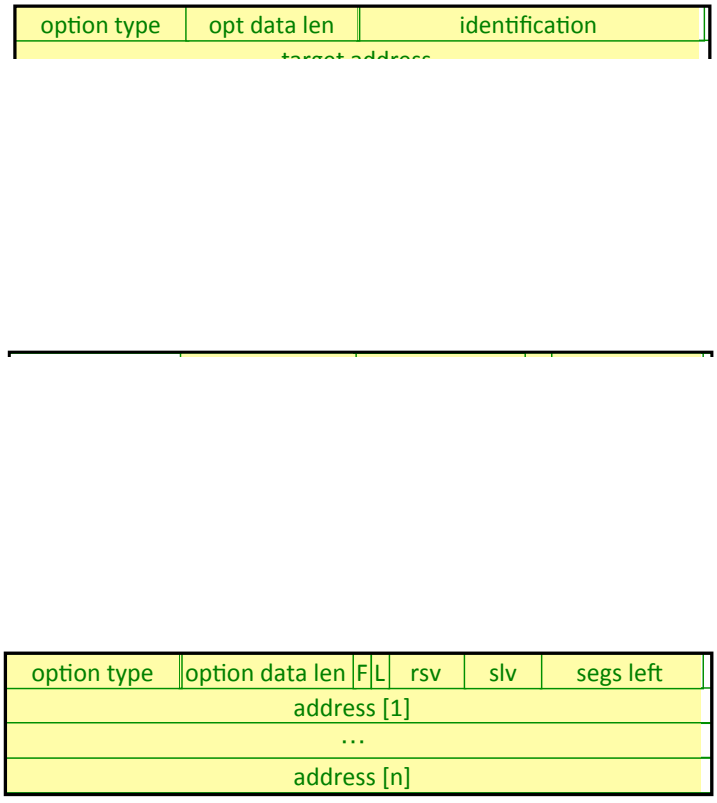

Figure 3.6. Packet format for `SourceRouteHeader`

3.1.1.2 Routing Functions

The routing functions of DSR consists of two parts: route discovery and route maintenance. We illustrates these two mechanisms in detail as follows.

Figure 3.8. Packet format for `AckRequestHeader`

The first routing function is the route discovery. When a node S has a packet to send to some destination node D but does not currently have a route to that node in its Route Cache, the node S saves the data packet in its Send Buffer and initiates route discovery process to find a route. To prevent from buffering the packets indefinitely, packets are dropped if they wait in the send buffer for more than `MaxSendBuffTime` (the default is 30 s). For route discovery, S transmits Route Request packets as local broadcast messages, specifying the target address and a unique request identifier. The node receiving the route request packet will check its identifier and target address in the request header, if the same one has been received before, the packet will be recognized as a duplicate and silently dropped. Otherwise, it appends its own node address to a list in the Route

Re ts

de of

th ly

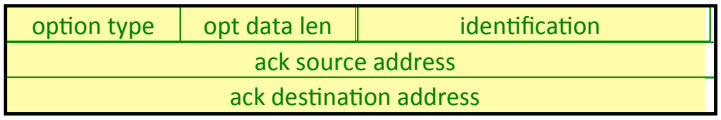

Figure 3.9. Packet format for `AckHeader`

header. When the Route Reply reaches the initiator of the request, it caches the new route in its Route Cache. Upon receiving the route reply message, node S will use the source route from Route Reply to send the data packet to D. Also, all the intermediate nodes receiving Route Reply packet will cut the route from their own to the de:

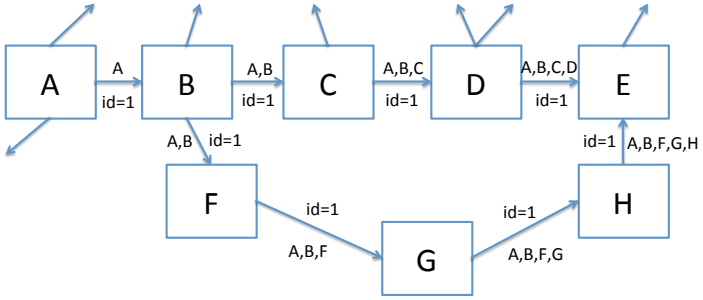

Figure 3.10. DSR route discovery mechanism

Another feature for Route Request packet is that when the intermediate node receives the packet, it searches the Route Cache to the destination address. If there is an existing route, the node attaches the route found to the route received from Route Request header, which forms a full route to the destination and it sends back to the source with the full route.

The second part of the route function is the route maintenance. Route maintenance is the mechanism by which the source node or any intermediate node is able to detect link breakage when the network topology has changed such that the source route in the Source Route header no longer works. This is a hop-by-hop operation and there are three mechanisms to verify the delivery of data packets to the next hop as discussed in Section 3.1.1.3. If the data packet fails to reach the next hop, the sender will retransmit the data packet. After `MaintenanceRetries` times of retransmission fails, a Route Error packet will be sent out. The packet will

not be dropped immediately, instead, a *salvage* mechanism will start by searching the route cache for alternative routes to the destination. If no alternative route is found, or the data packet has already been salvaged for MaxSalvageCount times, this data packet will be dropped.

DSR has a mechanism of removing stale route entries from the Route Cache of the node. If a node does not use the route for a period of time, that route entry will expire and be removed from the Route Cache. In our implementation, DSR waits for a RouteCacheTimeout before removing the entries, with a default value of 300 s.

3.1.1.3 Acknowledgment Mechanisms

There are three types of acknowledgement mechanisms specified in the RFC: link-layer, passive, and network-layer acknowledgement. If the media access control (MAC) protocol in use can provide feedback for the successful delivery of data packets, then it should then be used to notify the delivery of data packets and maintain the route validity. When link-layer acknowledgement is not available, but passive acknowledgement is, it should be used when the nodes can put themselves into promiscuous receive mode. When all the above two acknowledgement mechanisms are not available, the node should use network layer acknowledgements. When this mechanism is used, acknowledgement request header will be attached to the data packet before sending out. The acknowledgement will be sent back to the sender to notify the delivery of data packets.

3.1.1.4 DSR Optimization Mechanisms

Several optimizations have been specified in the DSR RFC: salvaging, gratuitous route reply, and increased spreading of route errors. The salvaging mechanism is triggered when the forward link is broken and there is an alternative route to the destination in the node's Route Cache. Instead of dropping the data packet, the node will try to retransmit it with the newly found route.

The gratuitous route reply is used when the node promiscuously received a data packet destined for other nodes but is named in the later unused portion of the packet's Source Route, then it can infer that the intermediate nodes before itself in the source route are no longer necessary, a gratuitous route reply packet will be sent back to the source with the shorter route.

The increased spreading of route errors means that when the node received the route error for the data packet it originated, it is the source of the data packet and the node will piggy-back the route error packet with the next route request process. This way it ensures that the route error packet spreads to the neighboring nodes and gets the expired route entries deleted.

Attribute	Defaults	Summary
MaxSendBuffLen	64	Maximum number of packets that can be stored in send buffer
MaxSendBuffTime	30 s	Maximum time packets can be queued in the send buffer
MaxMaintLen	50	Maximum number of packets that can be stored in maintenance buffer
MaxMaintTime	30 s	Maximum time packets can be queued in maintenance buffer
MaxCacheLen	64	Maximum number of route entries that can be stored in route cache
RouteCacheTimeout	300 s	Maximum time routes can be queued in route cache
SendBuffInterval	1 s	How often to check send buffer for pending data packets
NodeTraversalTime	100 ms	The time it takes to traverse the two neighboring nodes
MaxMaintRexmt	2	Maximum number of retransmissions for data packets
RreqRetries	16	Maximum number of retransmissions for route request discovery
MaintenanceRetries	2	Maximum number of retransmissions for data packets from maintenance buffer
RequestTableSize	64	Maximum number of request entries in the request table
RequestIdSize	16	Maximum number of request Ids in the request table
NonPropRequestTimeout	30 ms	The timeout value for non-propagation route requests
DiscoveryHopLimit	255	The maximum route discovery hop limit
MaxSalvageCount	16	The max salvage count for a single data packet
BlacklistTimeout	3 s	The time for a neighbor to stay in blacklist
GratReplyHoldoff	1 s	The time for gratuitous route reply entry to expire
BroadcastJitter	10 ms	The jitter time to avoid collision for broadcast packets
PassiveAckTimeout	100 ms	The time for a packet in maintenance buffer to wait for passive acknowledgment
RequestPeriod	500 ms	The base time interval between route requests
MaxRequestPeriod	10 s	The max time interval between route requests

Table 3.1. DSR attributes and their default values

3.1.1.5 DSR Data Structures

There are several conceptual data structures that are important to support DSR operation. Here we introduce the essential ones:

`RouteCache` contains all the routing information collected from the route discovery process. The structure of the DSR `RouteCache` is implemented as follows. Each entry implemented by the `RouteCacheEntry` class corresponds to a node in the network and the entry is mapped to that node's IP address. Every entry stores the following attributes of a node: the IP addresses from the source to the destination, which is saved in a vector, and the destination address of the route. Also, we store a time-stamp of the route entry when it is initially saved. The `RouteCache` will save multiple route cache entries for a single destination since DSR accepts multiple route replies. All the route cache entries for a single destination are saved according to the hop-count and freshness of the route. The route with least hop count will be saved before others, and for those routes with same hop count, the route entry that is newly found will be saved before other routes. The `RouteCache` class has implemented functions to add, delete, update, look up, and print entries. Also, unlike other routing protocols, DSR saves the whole source route in the route cache, and indexed only with the final destination. Therefore, when the direct route to a certain destination is not found, the look up route function we implemented will also search all the intermediate nodes in every single route entry for the expected destination. If the originating node found any intermediate node that matches the expected destination, the newly found route will be removed from the original route and indexed with the new destination address for future use. All the route cache entries are governed by a global timeout value `RouteCacheTimeout`, with default value of 300 s. The route

cache will be purged periodically to get rid of outdated routes.

`PacketBuffer` is used to save data packet whenever there is no route or the node is waiting for next hop delivery notification. There are two types of packet buffers, which are `Send Buffer` and `Maintenance Buffer`. These two buffers both queue data packets, the only difference being the `Send Buffer` will save the packet when receiving it from transport layer and when a route is not yet found, while the packet will be saved in the `Maintenance Buffer` when sending out from the `Send Buffer` yet waiting for the next hop acknowledgment. When DSR receives the data packet from transport protocol, it first checks the `RouteCache` for previously found route entries. The data packet is then saved to the `Send Buffer` if no existing routes are found, and the node will initiate route discovery process by broadcasting `Route Request` packets. When the packet in the `Send Buffer` is sent out with a `Source Route` header, the data packet will be saved in the `Maintenance Buffer` waiting for next hop delivery notification from the acknowledgement mechanism used. After a value of `MaxMaintenanceRetries` times retransmission with no delivery notifications, the data packet is removed from the `Maintenance Buffer`.

`RreqTable` is used to save route request information. It keeps track of two parts of route request operations: route request initiated by the node itself and request received by this node, which are implemented as `RreqRequestEntry` and `RreqRequestId`, respectively. `RreqTableEntry` saves the route requests that have been initiated by this node itself. The fields of the route entries include the destination address which records the specific destination this node has initiated route requests to, the time-to-live (TTL) which allows the node to implement a series of mechanisms to limit the transmission hops of the route request packet,

and the request number which is used to record the number of consecutive route requests that have been sent for a certain destination. The other entry `RreqTable` records is `RreqTableId`. This entry keeps track of the route requests this node has received from other nodes, and drop duplicate route request packets. It is mapped to the source node that originate the route request. The detailed entries include a destination address which records the specific destination this request is requesting route for, and the identification field which keeps track of the identification number of the route request packet. If the two fields in the `RreqTableId` are the same for two entries, the request packet will be recognized as a duplicate one and should be dropped silently.

`GratuitousRouteReply` is used to limit the rate at which the node originates route reply to the same sender from which it overhears a packet that triggers gratuitous route reply. Its entry includes three fields, `ReplyTo` is the address to which the node originates a gratuitous route reply, `ReplyFrom` keeps track of the node from which this node overhears the packet triggered the gratuitous route reply, and the `GratuitousHoldoffTime` is the remaining time before the expiration of the entry.

3.2 DSR Module Evaluation

To verify and evaluate the performance of our DSR routing protocol implementation, we performed simulations using the current version of the simulator, ns-3.12.1. We investigate the DSR performance with varying number of traffic sources and different pause times for mobility models and compare to the other existing MANET routing protocols in ns-3: DSDV, OLSR, and AODV.

3.2.1 Performance Metrics

The performance metrics for the evaluation of DSR routing protocol are packet delivery ratio (PDR) defined as the number of packets received divided by the number of packets sent by the application, routing overhead which is the fraction of bytes used by the protocol to send the DSR control messages, and delay which is the time takes by a packet to reach the destination node's MAC protocol from the source node's MAC protocol.

3.2.2 Simulation Setup

We performed the simulations over an area of 1500 $\times$ 300 m^2. All the simulations were averaged over 10 runs with each simulation running for 1000 s. Some of them are averaged over 20 runs to increase confidence. Simulations were performed with 50 nodes, and the source-sink traffic pair varies from 10, 20 and 30 nodes. The communication model is peer-to-peer communication with as many bidirectional flows as the number of nodes in the network. We performed simulations with a packet size of 64 bytes to exclude potential network congestion caused by large packets. All the nodes are configured to send 4 packets/s. Using this lower packet rate, we can accurately evaluate the performance of the routing protocols. We used the ns-3 `On-Off` application to generate CBR (constant bit rate) traffic. The 802.11b MAC is the link layer over Friis propagation loss model to limit the transmission ranges of nodes. The transmission range of the nodes is set at 250 m for evaluation. To achieve this transmission range, the transmit power was set to 8.9048 dBm. The mobility model used is random waypoint with random velocities from 0 – 20 m/s and pause times of 0 – 900 s.

DSR has several parameters with some of them interconnected with each other

and most of them are prone to change with different simulation scenarios like mobility model, node velocities, and node density. When the node velocity increases, the `RouteCacheTimeout` needs to decrease to get rid of invalid routes. Also, the `NodeTraversalTime` is the time for a packet to traverse the transmission range. This requires careful consideration since it directly affects the time to detect link breakage. If set too small, there will be possibility of false assumption of undelivered packets, while if too large, it takes too long to respond to link breakages. Thus, proper choice of `RouteCacheTimeout` and `NodeTraversalTime` is important in different simulation scenarios. DSR protocol parameters are highlighted in Table 4.6. All of the parameters are tuned to fit this specific scenario. The `NodeTraversalTime` is set as 2 μs to fit the 250 m in this case. RouteCacheTimeout is set as 300 s since this is a a case with relatively low mobility nodes.

Table 3.2 highlights the general simulation parameters used for performing these simulations. All simulations are performed with a total simulation time of 900 s, with a warm-up time of 50 s, which is set to make sure the mobility models can reach a steady-state. Constant-bit-rate (CBR) traffic is sent from 50 s to 1000 s. The simulation area is set as rectangular instead of square is to force the routing protocol to form routes with more hops, while at the same time being able to have a diversity of both long and short routes. Each case is run 10 times to make sure the accuracy of simulations. We use 64 bytes packet instead of large packet size because that the larger data packet can introduce network congestion much easier than small packets, and we want to avoid that when testing the performance of routing protocols.

Simulation Parameters	Value
Simulation Area	1500 m × 300 m
Number of runs	10
Warmup time	100 s
Total Simulation Time	1000 s
Mobility Model	Random way point
Node speed	0 - 20 m/s
Packet Size	64 bytes
Packet Rate	4 packets/s
Link layer	wifib-11mbs
RTS/CTS	No
Propagation Loss Model	Friis

Table 3.2. Simulation Parameters

Parameter	Values
RouteCacheTimeout	300 s
NodeTraversalTime	2 μs
MaxSendBuffLen	64
MaxSendBuffTime	30 s
MaxMaintLen	50
MaxMaintTime	30 s
PassiveAckTimeout	4 μs

Table 3.3. DSR parameters

3.2.3 Simulation Analysis

In the first scenario, we varied the pause time in Random WayPoint mobility model so that we can analyse the performance of DSR in both mobile and static scenarios. Figure 3.11 shows the variation of PDR by varying the pause time.

We see that as the number of nodes is increased, the packet delivery ratio also increases. However, PDR for 20 nodes is greater than that for 30 nodes for all pause times. This is due to the fact that as node density increases, the routing overhead also increases and this leads to more collisions in the network. Also, if we note the 95% confidence-interval error bars in Figure 3.11, as the pause time increases, so does the variation of PDR. This can be attributed to how the nodes

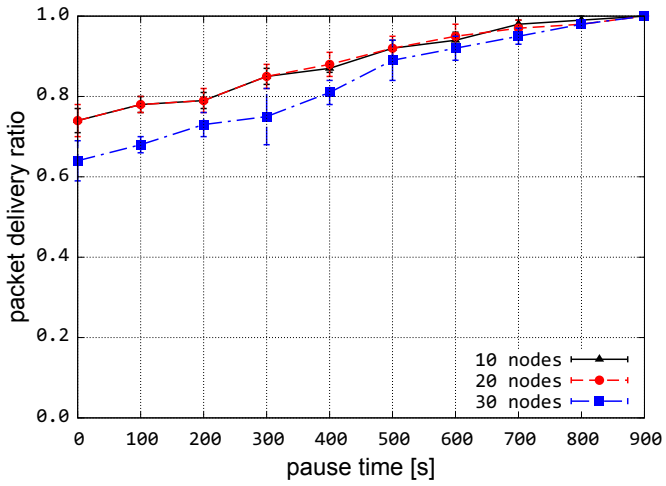

Figure 3.11. PDR with varying pause time

were positioned in the network initially since very long pause times will reduce movement from the initial position.

The routing overhead for different node densities with varying pause times is shown in Figure 3.12. This plot shows that overhead increases with the number of nodes. This is expected for DSR since as the node number increases, more route discovery packets will be sent out. The overhead for 30 nodes case has high overhead when the pause time is 0 s since as nodes moving constantly, route error packets will also be sent out more often.

We also considered the packet delay for data packets between source and destination. Figure 3.13 shows the delay decrease as the pause time increases. This is because as the pause time is increased the nodes will be immobile for longer durations and less route breakage will happen and less number of route discovery cycles will be needed. A slight increase in delay is observed with the source node number increases, since the route discovery overhead will increase when node number increases which increase the possibility of data packets being delayed.

In Figure 3.14, we compare the packet delivery ratio of existing MANET rout-

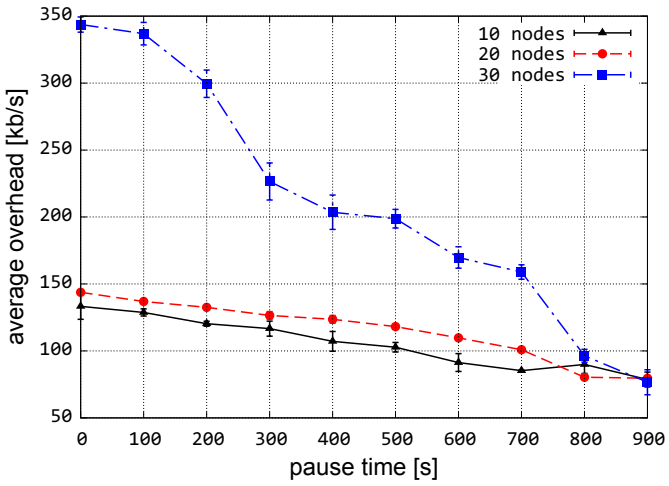

Figure 3.12. Overhead with varying pause time

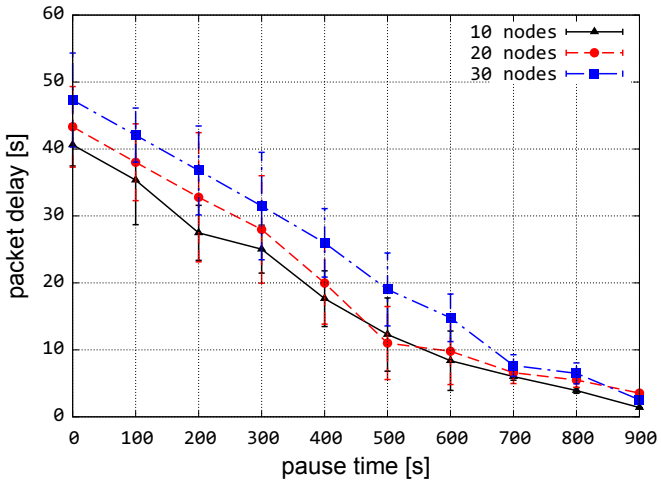

Figure 3.13. Packet delay with varying pause time

ing protocols implemented in ns-3 with DSR. From the plot we can see that OLSR outperforms DSR, DSDV, and AODV. The performance of DSDV follows the trend from the similar scenario in [2], while the performance of DSR and AODV has a little decrease compared to the result they have. This should be caused by the different MAC module they have used.

In our analysis, we also compared the routing overhead involved with all four

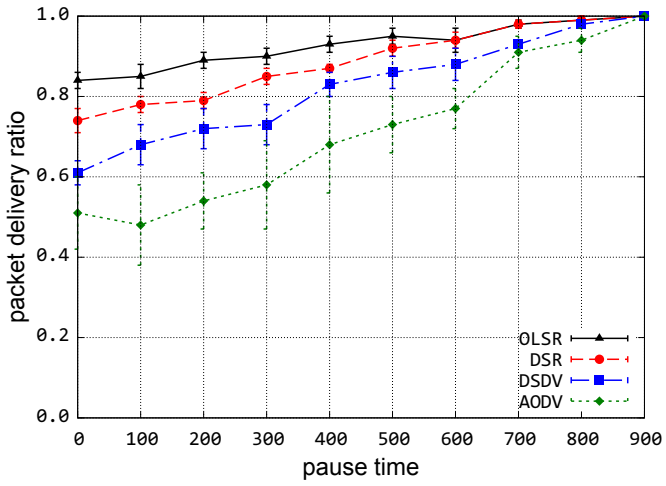

Figure 3.14. PDR with varying pause time

protocols. DSDV, DSR, and AODV have less overhead as the node number decreases since less number of control packets needed when nodes stay relatively immobile as shown in Figure 3.15. However, the overhead for OLSR increases with the pause time increase. This happens because as pause time increases, the initial distribution of nodes in the network determines the connectivity and the multipoint relays (MPRs) selection in OLSR. The overhead for DSR is slightly less than that for AODV since DSR does not need periodic broadcast packet like *Hello* packets which AODV uses.

We analysed the packet delay for these protocols and the result is shown in Figure 3.16. The packet delay is greater for DSR and AODV compared to that for DSDV and OLSR. This is because as reactive routing protocols, both DSR and AODV need more time to react to link changes while nodes moving. Also, we note that the variation of delay (shown as the error bar) for both DSR and AODV is comparably larger since they both rely on buffering extensively while in the route discovery cycle. As the pause time increases, the delay for all the protocols decrease since less link breakage happens.

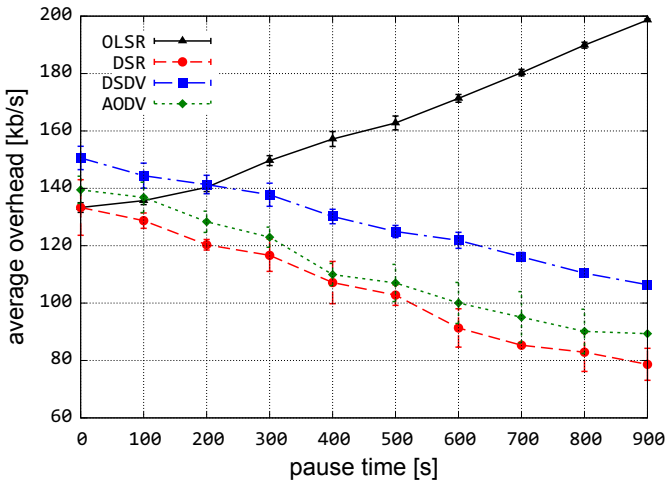

Figure 3.15. Overhead with varying pause time

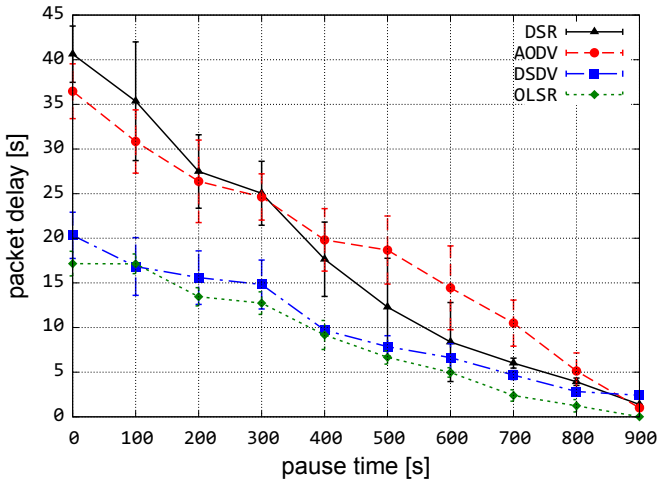

Figure 3.16. Packet delay with varying pause time

3.2.3.1 Performance Analysis of High Mobility Case

We keep all the general simulation variables the same while increasing the velocity of nodes. To cope with the high velocity, the DSR parameters need to be changed accordingly. As shown in table 3.4, we decreased the RouteCacheTimeout from 300 s to 30 s.

Table 3.4. DSR parameters for high mobility case

Parameter	Value
RouteCacheTimeout	30 s
NodeTraversalTime	2 μs
MaxSendBuffLen	100
MaxSendBuffTime	50 s
MaxMaintLen	100
MaxMaintTime	50 s
PassiveAckTimeout	4 μs

3.3 Implementation of Transactional Traffic Generator

A transactional traffic generator is an essential part for network simulation of the Internet. It is responsible for injecting synthetic traffic into the simulation according to a model of how application users would behave in certain circumstances. As the major contributor of transactional traffic, Hypertext Transfer Protocol (HTTP) [30, 31] is a pervasive application protocol and consumes a significant share of application flow in the Internet. Web traffic has transformed from plain-text Web pages to large size pages with embedded objects. An HTTP traffic model is needed to accurately represent and simulate Web traffic with the sustaining influence of HTTP over the Web. The HTTP protocol is now the de facto content delivery protocol, analysing its unique characteristics as well as its requirement on the network systems is required. However, to be able to develop an accurate HTTP traffic generator, we must first understand Web traffic characteristics. In the following section, we introduce the Web traffic features and the methodologies we use to develop our HTTP generator.

Based on the *connection-based* model [38], the PackMime-HTTP model in the ns-2 network simulator has been validated at packet-level by comparing synthetic traffic generated from simulation with measured. However, the model is problem-

atic in that it treats a collection of clients as a single client and a collection of servers as a single server. For example, a campus network that contains 100s of clients and tens of servers will be abstracted to one server and one client in this model. This is a useful simplification in the case of wired network since it can easily replicate a campus local network connecting to the Internet. However, when simulating wireless network, especially after introducing mobility to the hosts, the simplification in this model is too restrictive.

Therefore, we use the *connection-based* model from the Pack-Mime-HTTP generator with one modification to take individual Web servers and clients into consideration. One node can act as either server or client based on the start-time configuration of the simulation. This is advantageous as we can simulate both wired and wireless networks, with or without node mobility. For example, we can install the HTTP traffic generator in some of the nodes in the network, while running other traffic types in the remaining nodes, which gives the traffic generator more flexibility in simulation process.

Table 3.5. HTTP model attributes and their default values

Attribute	Defaults	Summary
MaxSessions	10	Number of Web sessions for the http simulation
InternetMode	*true*	Working mode is Internet-like or user-defined
Persistent	*true*	Connection is using persistent or not
Pipelining	*true*	Connection is using pipelining or not
UserNumPages	2	User defined number of Web pages for each session
UserNumObjects	2	User defined number of Web objects for each page
UserServerDelay	0.1 s	User defined server delay time to send out the response
UserPageRequestGap	0.2 s	User defined request gap time between two adjacent requests
UserObjectRequestGap	0.01 s	User defined request gap time between two adjacent requests
UserRequestSize	100 B	User defined Web request size
UserResponseSize	2048 B	User defined Web response size

This section describes the traffic model in our implementation of the HTTP traffic generator. This model is able to generate HTTP 1.0 traffic as well as HTTP 1.1 traffic with *persistent connection* and *pipelining*. The only feature lacking for HTTP 1.1 is the parallel TCP connection option planned the next code release. All the major attributes used in this implementation are listed in Table 3.5. The relationships among all of the classes implemented for this model are shown in Figure 3.17. The two major operations of our model are *source variable generation* and *transactions handling*, which we will discuss later in this section.

The model is capable of generating both *Internet-like* traffic and *user-defined* traffic, which are the two working modes of this generator. The difference between them is how the source variables are provided. In the *Internet-like* mode, we generate each source variable automatically based on its relative stochastic distribution function. Furthermore, all the distributions are calculated to represent two real-world packet traces [32, 53]. By following the source variable generation module [38], this generator can simulate network traffic that replicates real-world Internet traffic.

The *user-defined* mode is designed for the users who want to generate specialised network traffic, in which all the source variables in this model can be provided as parameters. The mode is designed to run simulations with detailed scenario settings controlled by the users. For example, users can test how different Web object sizes affect the network performance by tuning them while fixing all the other parameters. The two working modes are designed to suit most of the traffic generation requirements.

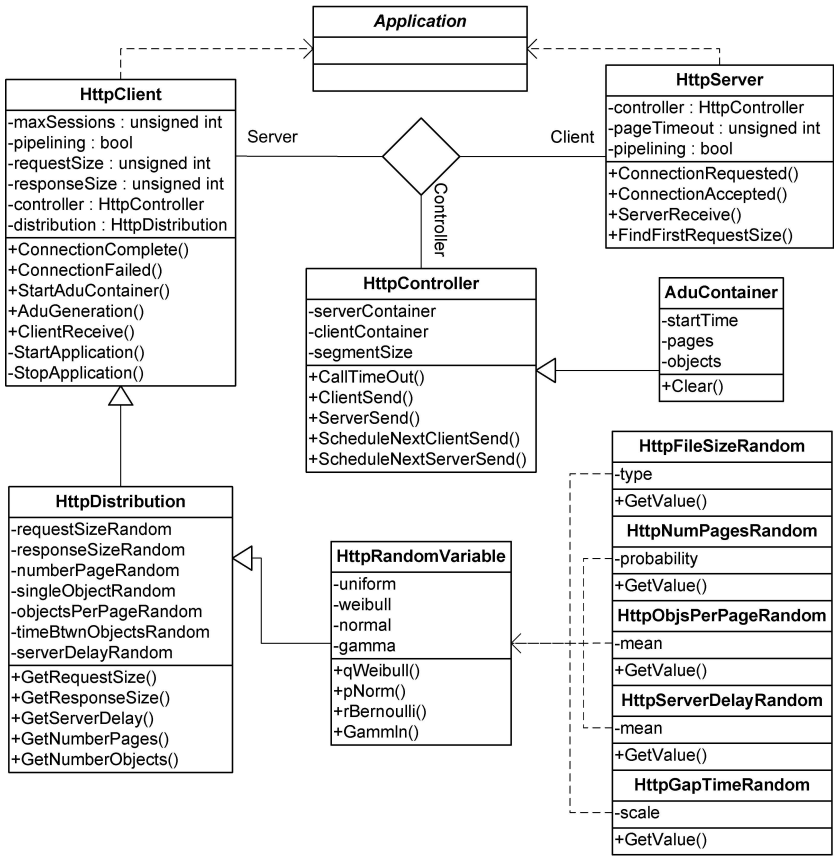

Figure 3.17. HTTP class diagram

3.3.1 Source Variable Generation

The source variable generation model is responsible for generating HTTP parameters for the *Internet-like* mode. MaxSessions is a user-defined value that specifies the number of Web sessions in the entire simulation process. The other major variables are: NumPages, ObjectsPerPage, ServerDelay,PageRequestGap, ObjectRequestGap, RequestSize, and ResponseSize. We use source variable generation functions to sample them, with each function for one of the parameters. NumPages and ObjectsPerPage are the number of pages for each Web session and the number of objects within one Web page, respectively. Both of them are modeled by the *Discrete Weibull* distribution [38] but with different distribution param-

eters. PageRequestGap is the inactive interval between two intermediate Web pages, while ObjectRequestGap is the interval for intermediate Web objects within a page. Both of them are fitted well by a combination of *Normal* and *Gamma* distributions [38]. ServerDelay is the time for the Web server to process the request, modeled by the *Inverse Weibull* distribution [38]. The maximum server delay is set as 10 s to avoid generating large delay values. For the same reason, we set the maximum request gap time as 100 ms.

Table 3.6. Transactional traffic model parameters

Parameters	Distributions	Model class
NumPages	Discrete Weibull	`HttpNumPages`
ObjectsPerPage	Discrete Weibull	`HttpObjsPerPage`
ServerDelay	Inverse Weibull	`HttpServerDelay`
PageRequestGap	Normal & Gamma	`HttpPageRequestGap`
ObjectRequestGap	Normal & Gamma	`HttpObjectRequestGap`
RequestSize	Discrete Weibull	`HttpFileSize`
ResponseSize	Discrete Weibull	`HttpFileSize`

Both distributions for RequestSize and ResponseSize are implemented in the `HttpFileSize` class, and they are fitted by *Discrete Weibull* distributions [38]. The mean of ResponseSize is larger than that of RequestSize. We assume the parameters in our model are uncorrelated with one another so that the generation of one parameter is independent of the other ones; this assumption has been verified with experiments in previous work [38,46]. The distribution functions for all the variables are shown in Table 3.3.2.

As shown in Figure 3.17, the `HttpClient` and `HttpServer` applications are responsible for the major functionalities, such as generating the transactional traffic, handling transactional processes, as well as recording results. When the HTTP model starts, the `HttpClient` and `HttpServer` applications are installed in client and server nodes, respectively. The `HttpClient` application starts when a

new TCP connection is initiated. On the other hand, the `HttpServer` application starts from the beginning of the simulation. The `HttpController` class controls the source variable generations and schedules the sending events for both the client and server. The user can define the number of clients and servers, not restricted to one client interacting with one server. In other words, each client can communicate with multiple servers, and one server can respond to multiple clients.

`HttpClient` starts by running `HttpDistribution`, which we develop for generating HTTP parameters. This implementation of our HTTP traffic generator provides several ns-3 `RandomVariable` objects for specifying distributions of HTTP source variables. It is based on the source code provided by PackMime-HTTP in ns-2 [11] and modified to fit into the ns-3 `RandomVariable` framework. `HttpRandomVariable` includes several subclasses with each one responsible for sampling one variable. For each Web session, the client first samples the number of objects for a specific TCP connection from the distribution of `HttpNumPages` and `HttpObjsPerPage` and sums up all the objects for the pages in each Web session. There are two gap times including `HttpObjectRequestGap` and `HttpPageRequestGap` as mentioned before. For each one of the requests, the client node also samples the HTTP request and response sizes based on `HttpFileSize`.

3.3.2 Transactions Handling

There are two types of application data units (ADUs) in this generator, `Request Adu` and `ResponseAdu`. After generating all the necessary source variables for both the `HttpClient` and `HttpServer` as we mentioned previously, this model saves the generated parameters into the respective ADUs. For example, the

`HttpObjectRequestGap` and `HttpPageRequestGap` will be saved in the `RequestAdu`, while the `HttpServerDelay` be saved in `ResponseAdu`. The `RequestAdu` and `ResponseAdu` are saved to the `AduContainer` in sequence. After saving all the ADUs in `AduContainer`, we make two copies and give both client and server one copy. The reason for doing this is to keep track of both the sending and receiving events. For example, when the server receives 1500 B of the Web request, it knows which response corresponded to this request and sends the correct response back to the client. The `HttpController` is responsible for managing the two ADU containers and scheduling the data sending events. The `AduContainer` class is designed to work with both HTTP 1.0 and HTTP 1.1, with the latter having either *persistent connection* and/or *pipelining*. When each ADU is sent out from the client or server, `HttpController` removes it from the corresponding `AduContainer` and continues with the next ADU until the container for both sides are empty, which notifies the end of one Web session. The model continues with the next Web session.

As described before, each side of the node pair will be installed with either the client or server application. The server starts from the beginning of the simulation and listens for request ADUs from the associated clients, while the client starts when one transport connection is established. The client first checks the HTTP version defined by the user. If it is HTTP 1.0, the client sends the next request only after receiving the response for the previous request; for an HTTP 1.1 connection, the client also samples the inter-request gap times based on `HttpGapTime` and sends requests after the gap time without waiting for the previous responses. The model repeats this process until all the requested ADUs are sent or have a timeout without receiving any responses. This timeout value is defined as `PageTimeout`

and it is a user tunable parameter. On the server side, when a `RequestAdu` arrives, the server locates it in server `AduContainer`. If the model finds one match, the Web request is deleted from the server `AduContainer` and the corresponding `ResponseAdu` is sent to the client after ServerDelay. This process will repeat until the requests are exhausted and all the responses are received by the client, following the next Web session. When all the Web sessions are finished transferring, the transport connection is closed and simulation is ended.

The result recording logic is triggered when one Web page is fully received. We consider the *object delivery ratio* and the *response latency* for each Web page as the performance metrics. If one Web page is not timed out, the *object delivery ratio* should always be one when using a reliable transport protocol such as TCP. The response latency is the time when client sent out the first request until the last byte of response for this specific Web page has been received. We have implemented our own result tracing system independent of the ns-3 built-in, which is included in our HTTP distribution. The reason behind this choice that for transactional traffic, users care more about the latency as well as the delivery ratio of the Web pages. It would be really difficult, if not impossible, to add tracing events in ns-3 notifying when a Web page is finished since the tracing events can only get information about how many TCP segments have been received by the node. There is no way to distinguish between two intermediate Web objects. This means that the result tracing logic is highly dependent on the HTTP model itself. When each page has finished receiving, we record how long it takes to successfully transfer the whole Web page. If TCP has timed out, we record what percentage of the Web page has been successfully delivered. All the tracing results are saved in different files and available to use after the simulation.

In this section, we present the simulation results conducted with the ns-3 network simulator [15] to analyse the performance of HTTP in different network conditions. Although the network characteristics data from [42] is aging, we still use it because this is the most detailed data public available and verified. Furthermore, this paper provides baseline results to compare against.

Several recent works have proposed Web traffic models based on current Web traffic data. One paper analyses Web traffic from 2006 to 2010 [70], capturing browsing behavior from more than 100 countries. Another recent paper [71] proposes their Web traffic models based on the top one million visited Web pages. We plan to test these datasets and models and may incorporate them into a future version of our ns-3 HTTP model if verified.

The model consists of three major parts: the distribution function generation model, the client application, and the server application. The distribution function generation model is responsible for generating HTTP parameters for simulation as shown in Table 3.3.2. The major parameters are: number of web sessions, number of web pages, number of inline objects in each page, inter-request time, and object size. The number of web sessions is the total number of HTTP sessions in the simulation. The total number of objects is the number of web pages multiply the number of inline objects in each page. It is fitted by the *Gamma* distribution [38, 46]. The inter-request time is the inactive interval between HTTP requests, and is fitted well by a heavy-tailed *Weibull* distribution [46]. The object size parameter consists of request and response sizes and both of them are fitted by a *Lognormal* distribution [38, 46]. The mean response size is mostly larger than request size.

The client and server applications are responsible for generating the transactional traffic over the entire simulation process. The client application controls

Parameters	Distributions	Model class
Web sessions	User-defined	User-defined
Request size	Lognormal	HTTPFileSize
Response size	Lognormal	HTTPFileSize
Total object number	Gamma	HTTPNumPage
	Gamma	HTTPObjsPerPage
Inter-request time	Weibull	HTTPXmitRate

Table 3.7. Transactional traffic model parameters

the parameters used in the simulation process, and is started when a new transport connection is initiated. On the other hand, the server application starts from the beginning of the simulation. The HTTP simulation workflow is illustrated in Figure 3.18. When the HTTP model starts, the client and server applications are installed in client and server nodes respectively. The users can define the number of clients and tiple servers, while one serv

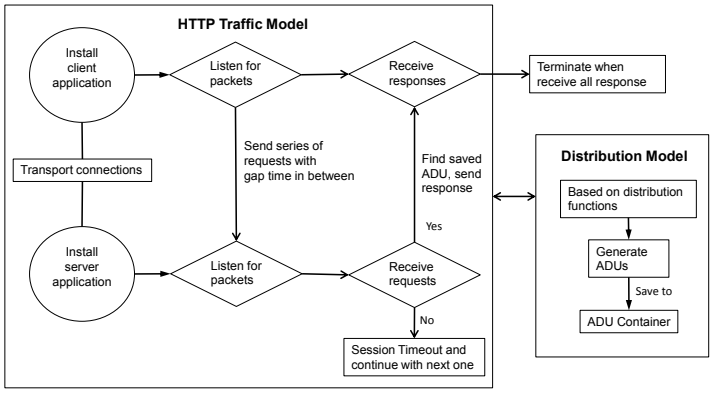

Figure 3.18. HTTP flow chart

Each HTTP client starts by running `HTTPRandomVariable`s, which we developed for generating HTTP parameters. This implementation of HTTP traffic generator provides several ns-3 `RandomVariable` objects for specifying distributions of HTTP connection variables. The implementations are taken from the

source code provided by PackMimeHTTP in ns-2 [11] and modified to fit into the ns-3 RandomVariable framework.

HTTPRandomVariable includes several subclasses with each one responsible for sampling a single parameter. For each web session, the client first samples the number of objects for a specific TCP connection from the distribution of HTTPNumPage and HTTPObjsPerPage and sums up all the objects for all the pages. The client then checks the HTTP version defined by the user. If it is HTTP 1.0, it sends the next request only after receiving the response for the previous request; for an HTTP 1.1 connection, the client also samples the inter-request times based on HTTPTXmitRate and sends requests without waiting for the previous responses. For each of the requests, the client node also samples the HTTP request and response sizes based on HTTPFileSize. The model saves the parameters generated to the RuntimeVariable table indexed with request size, globally accessible by all clients and servers, so they can synchronize with the parameters they use. The client sends the first HTTP request to the server and goes into the Listen state to wait for data transferred from the server. Once the client receives data from the server, it sets a timer to schedule the next HTTP transaction and repeats the process until all the transactions are done or a timeout happens without receiving any packets.

The server starts from the beginning of the simulation and listens for requests from the associated clients. When a request arrives, the server locates the RuntimeVariable entry as shown in Figure 3.18, indexed with the request file size, and retrieves the response size and the server-delay time saved in the entry. After this, the model schedules HTTP response traffic with the response size queried in the entry. The above process repeats until the requests are exhausted, and the TCP

connection is closed.

There are two working modes with the HTTP traffic generator, *automatic* and *manual* mode. The difference between the two modes is how the source variables are generated. In the automatic mode, we generate every source variable automatically based on its relative stochastic distribution. All the users need to define is how many web sessions a single simulation need to have. All the distributions in the model are calculated to represent two web traffic traces [32,53]. In the manual mode, users can change all the source variables in command line which is suitable when running simulations with detailed scenario settings. The two working modes are designed to suit the traffic generation uses.

The HTTP traffic generator we currently have can generate synthetic HTTP traffic with stochastic parameters. To generate web traffic with routers and switches, we model the TCP connections in terms of *source variables* including connection rate, startup overhead, the timing of request and response messages, as well as the file sizes transferred. This modeling process is named *connection-based* [38].

With pipelining, multiple HTTP requests are sent on a single HTTP connection without waiting for the corresponding responses. In ns-3 implementation, we implement a cache to save the pipelined HTTP responses, while waiting for the previous web object to finish transmitting. If it is HTTP/1.0 traffic, then the model saves web request in the cache and when the previous request/respond cycle is finished, the next in-line request will be sent. For the persistent TCP connections, we keep the connection open for the whole HTTP transaction process.

Once empirical distributions of the individual parameters are obtained, we compare each distribution with different standard probability distributions and

select the best fit trace figure. The Cumulative Distribution Function (CDF) plot is used to test whether the data fit the model. If the model fits the data perfectly then the plotted points should lie on a straight line. For most of the cases, the best standard probability distribution is determined to be the one that minimizes the root-mean-square of the deviation from a straight line. We select the best distribution from *Weibull*, *Lognormal*, *Gamma*, *Chi-square*, *Pareto* and *Exponential (Geometric)* distributions. (as shown in Table 3.3.2)

There are several essential HTTP parameters in the performance measurements: number of in-line objects, viewing time, in-line inter-arrival time, parsing time, and object size. We have a separate distribution generation model for each of the parameters. The number of in-line objects is always less than or equal to the total number of objects since a requested object does not need to be downloaded if it can be found in the web cache. Viewing time is the inactive interval between two intermediate HTTP requests. The distribution of viewing time is fitted well by a *heavy-tailed Weibull* distribution. In-line inter-arrival time is the time between the opening of one in-line object and the next one. The viewing time between the simultaneous objects is just a few tens of tens of milli seconds, while further in-line objects can be sent only upon completion of outstanding objects; thus, the inter-arrival times maybe a few seconds. The distribution for inter-arrival time of in-line objects matches a *Gamma* distribution. Parsing time is the time parsing the HTML code in order to determine the layout of a page. Since we do not model the detail layout of HTTP web pages, our HTTP traffic generator does not consider this time.

Both distribution of main-object size and in-line-object size are fitted by a *Lognormal* distribution. The mean of main-object size is larger than in-line-object

size; however, the variance of in-line-object size is greater than main-object size. The paper [46] tested the accuracy of this assumption with experiments. Therefore, we also assume the parameters in our model are assumed uncorrelated with themselves as well as with one another so that the generation of one parameter is independent of the generation of other parameters.

3.3.3 Transactional Traffic Model Validation

We verify that the source variable generation function can generate reasonable results compared to the distribution function described in previous work [38, 46, 49]. This is designed to test the operation of `HttpRandomVariable` in ns-3. We choose the `RequestSize`, `ResponseSize`, `RequestGap` and `ServerDelay` times as examples, and use a complementary cumulative distribution function (CCDF) to represent the results.

The Web object sizes include both request and response file sizes; the generation function for both is the *Discrete Weibull* distribution. However, the size of responses is significantly larger than that of requests. As we can see from Figure 3.19, for response sizes, 78% are smaller than 10 kB, and only 1% larger than 100 kB. While for request sizes, 97.5% are smaller than 1460 B, which would fit in one TCP segment. These two source variables follow the self-similar distribution as the number of large file sizes is small, while the number of small file sizes is large. This phenomena matches the real-world Internet traffic [43].

Two major time values in this model are the request gap time and server delay time as shown in Figure 3.20. The request gap time is the delay between two subsequent Web objects, while the server delay time is the latency for the server to process the ADU from clients and trigger the response sending mechanism.

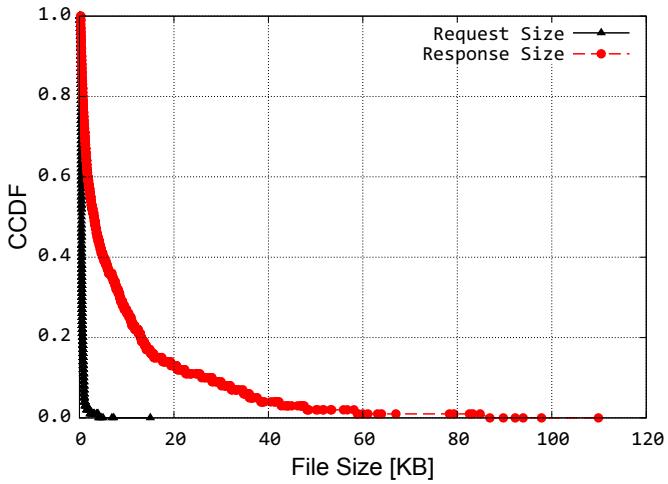

Figure 3.19. CCDF of HTTP file sizes

The request gap time follows a mixture of *Normal* and *Gamma* distributions [38]. Based on the generation functions for each time variable; 90% of the request gap time is below 10 ms, while 1% are larger than 25 ms. The server delay time follows an *Inverse Weibull* distribution, 90% are below 500 ms, with only 1% larger than 1000 ms. Both of the two time values follow self-similar distributions. The maximum request gap time is set as 100 ms, which explains the cutoff of the request gap time at that time value.

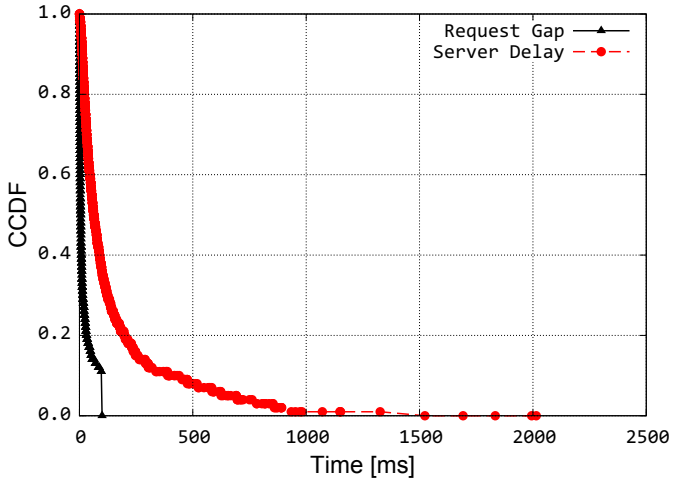

Figure 3.20. CCDF of HTTP delay times

Chapter 4

Simulation Analysis

We carry out a systematic performance comparison of transport protocols, routing protocols while using different types of application traffic in MANET environment. The transport protocols involve TCP [72], UDP [73], combined traffic, and HTTP traffic. The routing protocols involved for comparison are DSR [7, 57], DSDV [6, 19], AODV [5, 56], and OLSR [8]. Different types of application traffic is another facet we want to examine. We will compare constant bit rate (CBR), bulk data transfer, transactional traffic, and how the different traffic characteristics affect the network performance. The parameters varying will mainly be the node velocity and node density. The majority of this chapter has been published in WNS3 paper [22].

Different traffic types are supposed to have different network requirements and show different performance characteristics. Constant bit rate (CBR) traffic is used to test UDP traffic and is relatively well studied. Bulk data transfer is used to support TCP and transfer a relative large chunk of data from one end to the other. It is normally used to test how different transport protocols perform. The large transfer protocol in ns-3 is an FTP-style traffic generator and can transfer large

files and test how different transport protocols perform when carrying large file from one end node to the other one. Transactional traffic is gaining importance in wireless network. As part of my research, I implemented the HTTP traffic generator to generate transactional traffic and DSR routing protocol and tested it under various scenarios.

The traffic model we used is different from the previous simulation scenarios for highly-dynamic environment [66, 74–77]. The previous model has one ground station in the middle and serve as the global sink with the test articles moving around it. For this thesis, we manage to have a fair comparison among all the routing and transport protocols, and to have test scenarios suitable for all speed cases. The traffic model we use is the normal MANET scenario sets, in which each node is transmitting to each other. With this traffic model, there are more traffic flows involved and presumably more routing overhead involved since more data is transmitting throughout the network. The delay is expected to be higher due to the extra traffic that stacks up the queue. Furthermore, if the velocity of the nodes is very high, there are more link breakage and overhead. The overhead imposed by the routing protocol is measured in terms of both packets and bytes.

For the routing comparison part, the paper [2] compares DSR, DSDV, OLSR, and AODV in low mobility case with a maximum node speed at 20 m/s. We have tested similar cases and generated comparable results [21]. Besides these cases, we are also interested in testing the scenarios with highly-dynamic nodes which is the normal case in aeronautical environments. We plan to test the speed ranging from 20 m/s all the way up to 1200 m/s to have a good coverage of all cases. ResTP and AeroRP are designed for such highly-dynamic environment; therefore, we expect the combination of these two to perform much better than the other

transport and routing protocol combinations in highly-dynamic environment. The network performance can be affected by a lot of different factors and we would like to investigate how some of them interact with each other, and how they affect the network performance.

We plan to test different mobility models including steady-state random waypoint, random walk, random direction, and 3D Gauss Markov mobility model. The random waypoint mobility model is widely used in literature for routing protocol comparison [2]. However, it does not represent the real world movement and may create misleading simulation results [10, 69]. Furthermore, the random waypoint mobility model takes a lot of time to enter the stationary mode especially when the nodes are moving slowly [20]. Based on the analysis, we choose steady-state random waypoint and 3D Gauss Markov mobility model. The 3D Gauss Markov mobility model is designed to simulate the network traffic more realistically and does not make sharp turns so that it can represent the movement of aircrafts more realistically. Our ResiliNets group has implemented the 3D Gauss Markov mobility mode in ns-3 [20].

The study centers on investigating and quantifying the effects of various factors and their effects on wireless network performance. The paper isolates and quantifies the effects of five factors that affecting the performance of ad hoc networks including node speed, pause-time, network size, number of traffic sources, and types of routing mechanisms. The achievable capacity of ad hoc wireless networks depends on network size, traffic patterns, and detailed local radio interactions. We will vary the network size to see how different protocols evolve with the different network sizes. We vary the number of traffic flows over the network to see how different traffic loads affect network performance. The symptom of

failure under stress is congestion loss. Large number of nodes, routing queries, high mobility case together introduce a high volume of routing queries or updates caused network congestion. Evaluation of routing protocols tends to use low data rate to avoid the network congestion. The loads used in other ad hoc routing studies are consonant with our work; for example, both Karp, Kung and Broch et al. limit the total offered load to about 60 Kbps despite using 2 Mbps radios. We vary the traffic load to check the difference in performance and find the optimum transfer data rate for certain type of MANET networks.

For the packet size, we choose 64 bytes for the application packet size. This packet size choice is for the easy comparison of routing protocols for CBR traffic. When the packet size is small, the percentage of packet size compared to that of the control packet size is small, this way it is testing how the routing protocol performs in terms of routing protocols.

Table 4.1 highlights the general simulation parameters used for performing these simulations. All simulations are performed in ns-3-dev with a total simulation time of 1500 s. Constant bit-rate (CBR) traffic is sent from 100 s to 1000 s. A shutdown time of 400 s is set so that any packets that are buffered will be transmitted during this time. This makes sure that all the CBR packets sent by a source will have enough time to reach the destination and would be helpful for highly-dynamic environment.

The low mobility case is intended to compare with past comparison works. While there is no performance comparison work takes high mobility nodes into consideration, especially the highly-dynamic telemetry network, we would like to introduce high speed to test how different routing protocols and transport protocols perform when facing more link breakage introduced by fast moving

Parameter	Value
ns-3 version	ns-3-dev
Number of simulation counts	10
Simulation area	1500 m × 300 m
Initial position allocator	Random rectangle
Warmup time	100 s
Traffic time	900 s
Link layer	802.11b DSSS 11Mbps
RTS/CTS?	No
Packet fragmentation?	No
Propagation loss model	Range
Transport protocol	TCP, UDP
Routing protocol	AODV, DSDV, DSR, and OLSR
Total node number	50
Sink number	10, 20, and 30 nodes
Mobility model	Steady-state Random WayPoint, 3D Gauss-Markov, RandomDirection, and RandomWalk
Low velocity	1 m/s, 20 m/s
Pause time	0, 100, 200, 300, 400, 500, 600, 700, 800, 900 s
High velocity	100 m/s, 200 m/s, 300 m/s, 400 m/s, 500 m/s, 600 m/s, 700 m/s, 800 m/s, 900 m/s, 1000 m/s

Table 4.1. General simulation parameters

Parameter	Value
Application types	HTTP, CBR, and bulk data transfer
CBR traffic	64 bytes with 4 pkts/s
HTTP traffic	HTTP sessions
Bulk data	1000 MB for one node pair

Table 4.2. Traffic Patterns

nodes.

All the OLSR routing protocol parameters are set to default values in ns-3-dev except for the ones highlighted in Table 4.3. HelloInterval is changed from a default of 5 s to 1 s, similarly, TcIInterval is set to 5 s and MidInterval to 5 s as well to suit the highly dynamic nature of this simulation environment.

Parameter	**Value**
HelloInterval	1 s
TcInterval	5 s
MidInterval	5 s

Table 4.3. OLSR parameters

Table 4.4 highlights the parameters chosen for AODV routing protocol. Similar to the way OLSR routing protocol, parameters are modified to suit the highly dynamic scenario, some of AODV's parameters are also modified. RreqRateLimit is changed from its default value of 10 to 5.

Parameter	**Value**
RreqRateLimit	5 RREQ per second
NextHopWait	50 ms
MyRouteTimeout	11.2 s
BlackListTimeout	5.6 s
DeletePeriod	8 s
NetDiameter	Number of nodes - 1
NetTraversalTime	2.8 s
PathDiscoveryTime	5.6 s

Table 4.4. AODV parameters

DSDV routing protocol parameters are highlighted in Table 4.5. ForwardingInterval is modified from its default value of 15 s to 4 s and the SettlingTime is changed from 5 s to 0 s. With a SettlingTime of 0 s, DSDV will use a route in

its perusal immediately without waiting to see if the route is stable or not. In addition, buffering is enabled in DSDV with a maximum buffer time set to 30 s.

Parameter	Value
SettlingTime	0 s
MaxQueueLen	Number of nodes × MaxQueuedPacketsPerDst
MaxQueuedPacketsPerDst	5 packets

Table 4.5. DSDV parameters

DSR routing protocol parameters are highlighted in Table 4.6. RouteCacheTimeout is modified from its default value of 300 s to 30 s. The NodeTraversalTime is the time the data packet takes to traverse from one node to the other, which is set as 2 ms. The PassiveAckTimeout doubles the value of NodeTraversalTime.

Parameter	Value
RouteCacheTimeout	300 s
NodeTraversalTime	2 ms
PassiveAckTimeout	4 ms
MaxSendBuffLen	64
MaxSendBuffTime	30 s
MaxMaintLen	50
MaxMaintTime	30 s
MaxCacheLen	64
MaintenanceRetries	3

Table 4.6. DSR parameters

4.1 Performance Metrics

The performance metrics considered for the evaluation of CBR traffic are packet delivery ratio (PDR), routing overhead, and delay.

- **Packet Delivery Ratio**: The number of packets received divided by the number of packets sent by the application.
- **Routing Overhead**: The fraction of bytes used by the protocol for control messages
- **Delay**: The time taken by the packet to reach the destination node's MAC protocol from the source node's MAC protocol.

For the bulk data transfer traffic, we choose the metrics according to its specific characteristics including overall throughput and delay.

- **Overall Throughput**: The number of unique data packets or bytes received at the sink divided by the total transfer time. This metric considers all phases and overhead.
- **Delay**: The time it takes to transfer certain chunk of data from one end to the other.

For transactional traffic, we use the same set of scenarios from CBR traffic. For the object size, we choose 512 B as the starting point to fit in a single packet since the TCP model in ns-3 uses 536 B as fragmentation threshold. We test the object sizes of 1 and 20 packets which are 512 B and 10240 B in size respectively. We also set the maximum number of web transactions at 500. This value is chosen to have a reasonable length of total simulation time. We use object delivery ratio (ODR) and user perceived delay (UPD) as the performance metrics while evaluating HTTP traffic over different routing protocols:

- **Object Delivery Ratio (ODR)** is the number of objects received divided by the number of objects sent from server to client

- **User Perceived Delay (UPD)** is the time from the object request been issued to the time when the whole object has been received by the client.

The reason we choose ODR and UPD instead of PDR and packet delay is that for HTTP traffic, we do not care too much about the delivery of a single packet, instead, the percentage of objects delivered is much more important for the Web experience. The same reason also applies for UPD since users usually do not care about the delay for a single packet; however, they just notice the loading delay of the web objects.

The performance of routing transactional traffic in MANETs with low mobility has been examined [27, 28], but there is no previous work that has considered transactional traffic over airborne networks with high velocity airborne nodes. We present the interaction of transactional traffic with AeroRP in a high-dynamic environment and compare the performance using legacy MANET routing protocols in the same set of scenarios. We use the ns-3 network simulator [15] to analyse the performance of transactional traffic in a set of environments with different characterists.

4.2 Simulations with CBR Traffic

We analysed the performance of different routing protocols in the simulation scenario we explained before with CBR application traffic, by taking use of the ns-3 OnOffApplication traffic class. The number of traffic flows is ten. All the nodes are configured to send 4 packet/s with a packet size of 64 B. For the CBR traffic, we consider the network performance metrics of PDR (packet delivery ratio), overhead, and delay. We start with the low mobility cases where the pause time ranges from 0 s to 900 s, and the speed is random picked from 0 m/s to

20 m/s.

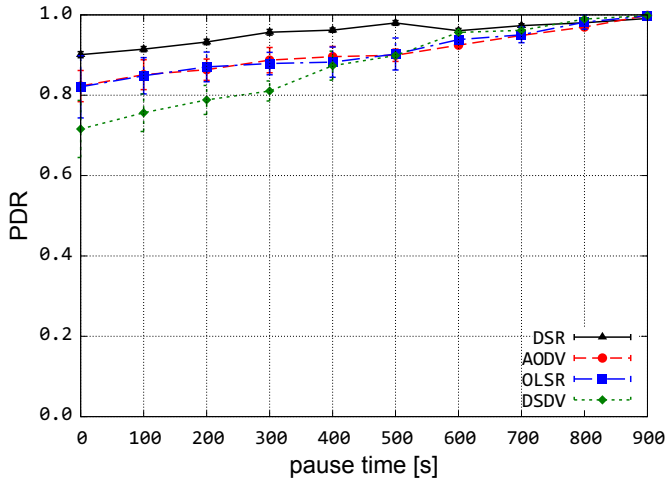

Figure 4.1. PDR of different pause time with CBR

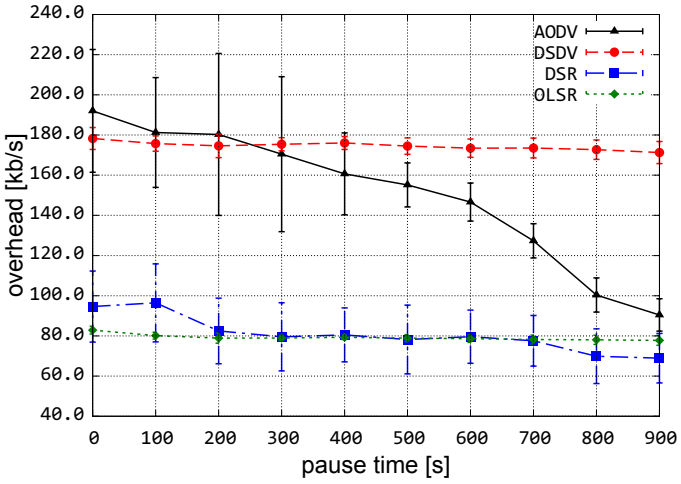

Figure 4.2. Overhead of different pause time with CBR

Figure 4.1 shows the average PDR as the pause time for Random Waypoint mobility model increases. As the pause time increases, the PDR increases because the node mobility decreases. When the pause time is 900, which means the nodes are static, the packet delivery ratio is 1. DSR performs the best among all the protocols compared and its performance matches the previous comparison work [2].

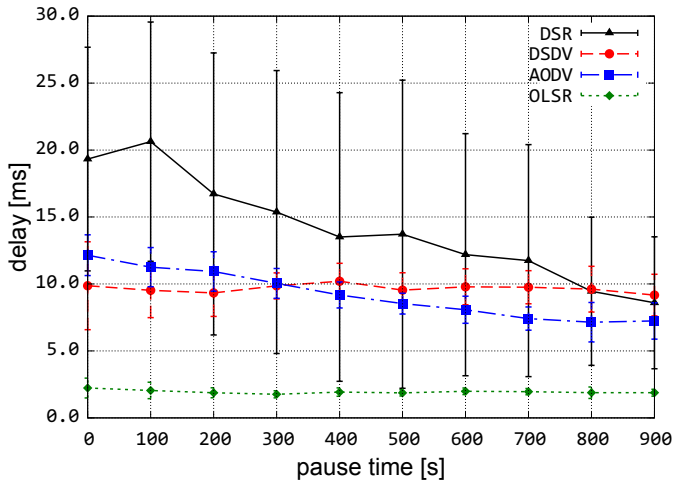

Figure 4.3. Delay of different pause time with CBR

DSDV performs the worst, when the pause time is zero, it does not converge well and has a packet delivery ratio of 72%. AODV and OLSR perform similar, with 80 percent packet delivery when the pause time is zero.

Figure 4.3 shows the average delay with the varying pause time. As the pause time increases, the delay decreases for the reactive routing protocols AODV and DSR. This is because that as the mobility of nodes decreases, less route breakages happen, so it takes less time for the packets stay at the queues. While at the same time for the proactive routing protocols, the delay varies less compared to that of the reactive routing protocols.

Figure 4.2 presents the average overhead with the varying pause time. As the pause time increases, the delay decreases for the reactive routing protocols AODV and DSR. This is because that as the moving speed of nodes decreases, less route breakages happen, and less route request messages needed to maintain the active routes. While at the same time for proactive routing protocols, they will keep sending topology information regardless of how the network topology changes.

4.2.1 Aeronautical environment

We have tested the aeronautical environment when there is a single sink for all the traffic sources, and the nodes are traveling at speed varying from 100 m/s to 1000 m/s. We introduced AeroRP [65] in this simulation comparison. We list the parameters for AeroRP in Table 4.7

Parameter	Value
`hello` beacon interval	1 s
Packet queue check interval	0.5 s
Neighbor hold time	4 s
Transmission range	27800 m
Max packet queue length	10000 packets
Max packet queue hold time	1000 s
GSUpdateInterval	20 s
GSTriggerUpdateInterval	5 s
Ferry sending rate	1.024 kb/s

Table 4.7. AeroRP parameters

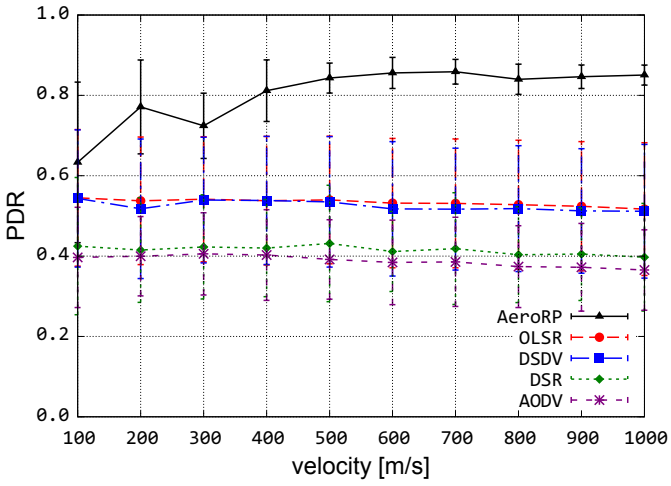

Figure 4.4. PDR of different velocity with CBR

Figure 4.4 shows the average PDR as the speed of the nodes increases in the aeronautical environment. As the speed increases, all the protocols except AeroRP

decrease in PDR. However, the decreasing is not very much. The reason is that all the other protocols all cache the packets when there is no active route for the packets. The PDR for AeroRP increases as the speed increases. This is because it makes forwarding decision per hop, and as the speed increases, the possibility that the packets will be contacting other nodes increases, which increases the packet delivery ratio.

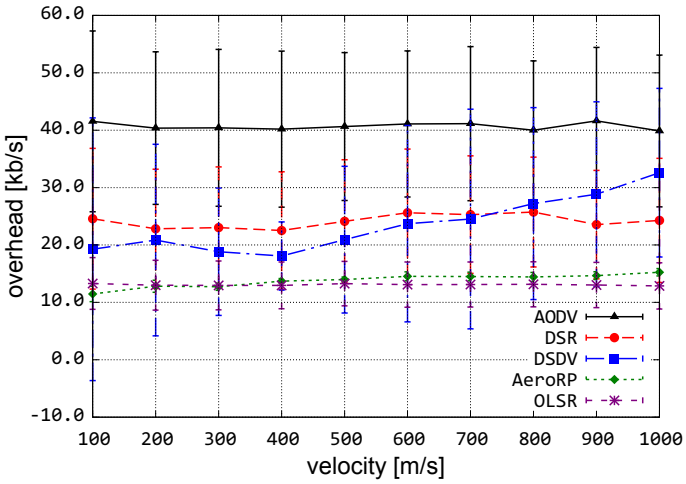

Figure 4.5. Overhead of velocity with CBR

Overhead of all the protocols is shown in Figure 4.5. AODV has the most overhead in the protocols following DSR. This is because in the high mobility cases, AODV and DSR need to broadcast route requests constantly to find a new route when the previous one breaks, which is quite frequent in high velocity scenario. While DSDV, OLSR, and AeroRP have relatively less overhead. DSDV increases a little in terms of overhead as the speed increases, this is because DSDV has one topology update mechanism that is incremental update, and it generates more control packet when the route breaks become more frequent.

End-to-end delay for the compared protocols is presented in Figure 4.6. Delay for AeroRP is higher than the other routing protocols since this scenarios use ferry

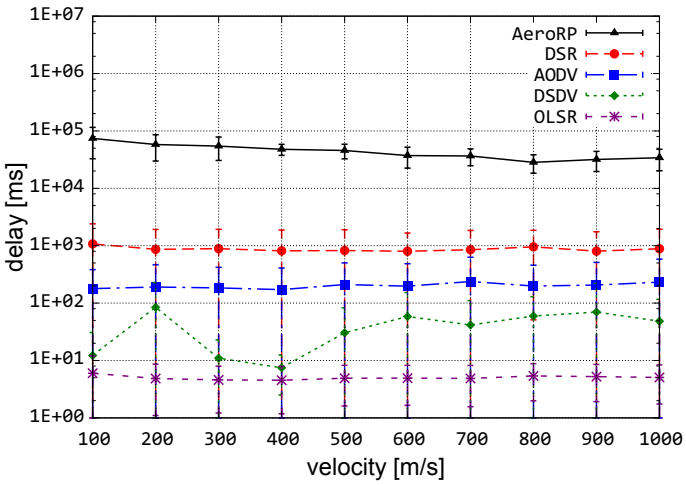

Figure 4.6. Delay of different velocity with CBR

mode of AeroRP, in which the packet cache stores all unsent packets. This makes the delay higher since the packets with no route stack up in the queue. Note that this is the cost for higher PDR in which these packets do eventually get delivered. DSR has the lowest delay around 50 ms. The delay for DSDV, OLSR, and DSR stays below 200 ms for all the scenarios mainly because they use a drop-tail queue with limited queue sizes.

4.3 Simulations with Bulk Data Traffic

We analysed the performance of different routing protocols in the highly-dynamic aeronautical environment with bulk data traffic, using the ns-3 BulkSendApplication traffic class.

Figure 4.7 shows the average goodput of all the protocols in different velocity cases. When the velocity is below 100 m/s, the goodput can reach up to 5 Mb/s, while when the velocity reaches 100 m/s, the goodput drops to before 1 Mb/s. DSDV and OLSR perform better than AODV and DSR. For example, when the

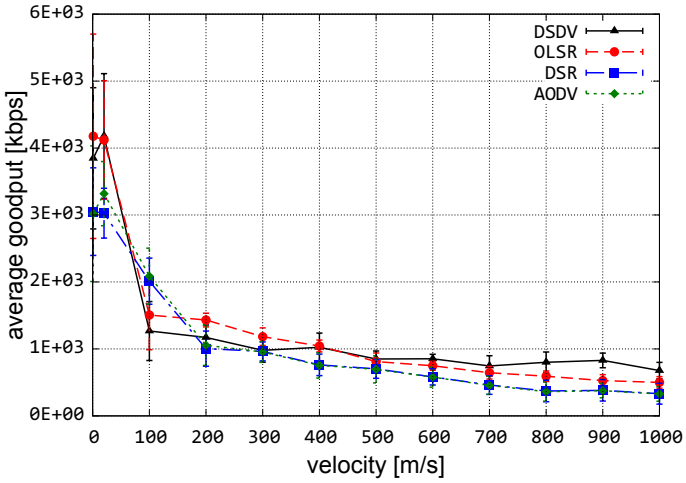

Figure 4.7. Average goodput of different velocity with bulk transfer

speed is 1000 m/s, goodput for DSDV is about 250 Kb/s better than that of DSR.

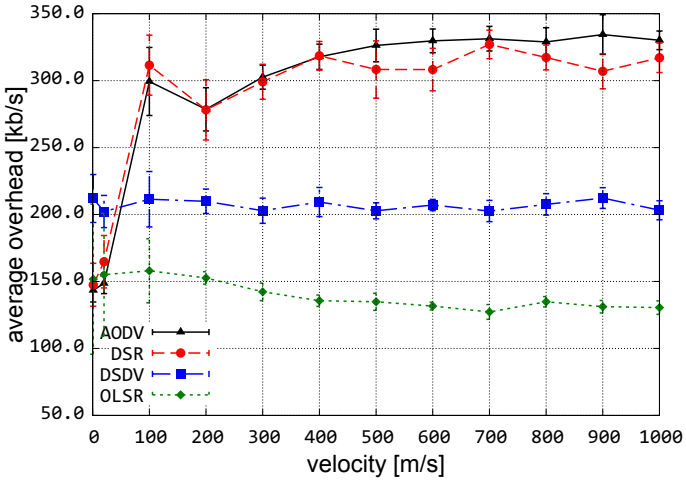

Figure 4.8. Average overhead of velocity with bulk transfer

Figure 4.8 presents the overhead involved in the protocols. The overhead for both AODV and DSR increases as the velocity increases. This is because as the nodes move faster, the more link breakages happen, and then more route discovery process needs to be initiated. While the overhead for both DSDV and OLSR stays pretty much the same for all the velocity scenarios. This is because

they are proactive routing protocols, and do not change topology update behavior much due to the velocity changes.

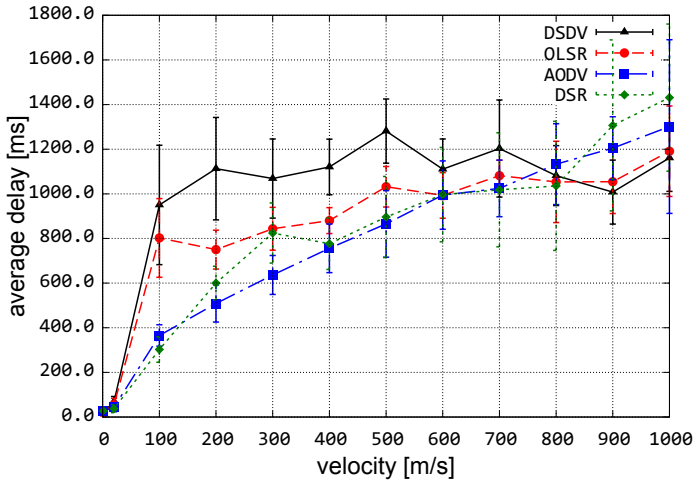

Figure 4.9. Average delay of different velocity with bulk transfer

Figure 4.9 shows the end-to-end delay of the protocols. As the speed of nodes increases, the delay increases and stabilises around 1 s. The difference between different routing protocols is not very significant when the velocity becomes higher than 100 m/s.

4.4 Transactional Traffic Performance

For transactional traffic, we use a different set of metrics and scenarios setting due to its different characteristics from UDP traffic. Previous work [42] has defined characteristics for different networks including maximum segment size (MSS), round trip time (RTT), and the bandwidth for network links as shown in Table 4.8. Some of the networks are from measurement of the actual systems. For example, a 10 Mb/s Ethernet connection was measured between two Sun hosts. For some of the other networks, parameters were estimated; for example, Mo-

dem and ISDN used theoretical bandwidths. The networks with N-Modem and N-F-Internet [78] represents two similar network characteristics with Modem and Fast-Internet, respectively. They are just the same networks with a slight different characteristics. We include them to verify our model. The networks with the HTTP workloads are used to test how different size of Web pages and number of request-response transactions affect HTTP performance and they are shown as follows [42]:

- **Small Page**: single 5 kB Web page
- **Medium Page**: single 25 kB Web page
- **Large Page**: single 100 kB Web page
- **Small Cluster**: single 6651 B page with embedded 3883 B and 1866 B images
- **Medium Cluster**: single 3220 B page with three embedded images of sizes 57613 B, 2344 B, and 14190 B
- **Large Cluster**: single 100 kB page with 10 embedded 25 kB images

In addition to these workloads, we also use a varying workload with a range of 10 to 1000 Web objects with the fixed object size 10 kB, to test different HTTP versions: pipelining, persistent, and non-persistent connection. We will show the results in Section 4.4.1.

Our validation scenario consists of two nodes, one server and one client, with a point-to-point link connecting them. The transport protocol in this case is TCP. We use *user-defined* mode of our generator and use the following variables:

Table 4.8. Network characteristics

Network	RTT (ms)	BW (Mb/s)	MSS (B)
Fast-Internet	89	1.02	512
N-F-Internet	80	1.17	1460
ADSL	30	6	512
Ethernet	0.7	8.72	1460
Fast-Ethernet	0.7	100	1460
Modem	250	0.0275	512
DirecPC	500	1	512
Slow-Internet	161	0.102	512
ISDN	30	0.122	512
WAN-Modem	350	0.0275	512
WAN-ISDN	130	0.122	512
N-Modem	150	0.0275	1460

- UserObjectRequestGap: 0.01 s
- UserServerDelay: 0.1 s
- UserPageRequestGap: 0.2 s
- UserPageRequestSize: 256 B

We only consider a wired scenario in this paper, with only one server-client pair analysed. We plan to perform more detailed wireless simulations later with varying number of server-client pairs, including based on our previous work simulating wireless scenarios [75]. We choose `UserObjectRequestGap`, `UserServerDelay`, and `UserServerDelay` based on the most frequent values in our distribution model, and use 256 B of request size that would fit in one TCP segment for most real-world MTUs. This way the latency is mainly dependent on the response size. For the simulation metric, we use the Web page response latency and do not present object delivery ratio since TCP guarantees the delivery of data segments, and all the cases we tested do have 100% object delivery ratio. We use

HTTP 1.1 with persistent connection and pipelining for all the simulation cases except for the last one in which we test how different HTTP versions would affect the response latency.

4.4.1 Simulation Results

The first simulation scenario is to test how different workloads affect network performance in terms of response latency. The result is shown in Table 4.9 with the time scale in milliseconds. As the Web page sizes increase, the latency for all the networks increases as expected. Although some of the values are not exactly the same as the theoretical values [42], they are of the same order of magnitude. Furthermore, we additionally take server processing delay into consideration, which is another factor that contributes to the overall latency. When plotting the performance of all the networks, we did not include the WAN-Modem and N-Modem curves because their performance are very similar to the Modem curve. For the same reason, we do not present WAN-ISDN here and show the ISDN curve.

Table 4.9. Latency of different networks [ms]

Network	Small page	Medium page	Large page	Small cluster	Medium cluster	Large cluster
Fast-Internet	477.5	819.6	1471.7	808.5	1456.3	3852.8
N-Fast-Internet	277.4	587.2	1123.0	510.3	1141.2	3158.0
ADSL	155.5	258.3	395.3	256.4	411.2	935.4
Ethernet	7.1	26.0	96.7	16.5	71.0	262.3
Fast-Ethernet	2.5	5.4	11.6	4.4	11.7	25.7
Modem	2941.3	9700.3	33314.6	6829.5	25468.8	91694.2
DirecPC	2546.5	7329.5	15125.6	5501.0	12969.6	27129.3
Slow-Internet	1293.4	3653.1	10019.7	2769.2	8185.2	26695.3
ISDN	465.7	1884.6	7207.5	1263.4	5359.9	20106.1
WAN-Modem	3441.3	10580.1	34194.4	7669.4	26648.6	93574.0
WAN-ISDN	922.2	2341.1	7664.03	1922.7	6116.4	21562.6
N-Modem	3466.9	9456.1	31897.5	6880.8	24191.6	86515.7

We further test how the response size would affect HTTP performance. We use the same network configuration as previous simulations. As the response size increases, the response latency increases for both of the fast network curves in Figure 4.10 as well as the slow networks in Figure 4.11. While for the slow network, the increase degree is larger, they are more sensitive to the increasing size of responses. For example, as shown in Figure 4.11, the latency of Modem increases from 15 s to 33 s when response size increases from 40 kB to 100 kB; however the latency only increases from 10 s to 15 s for DirecPC.

The latency for the fast networks in Figure 4.10 is within the delay range tolerable for everyday use. We can see that for Fast-Internet, the latency increases to 1.5 s when the response size is 100 kB. The latency is a little bit larger than we normally desire in current Web browsing. This is because all the network data from [42] is estimated 10 years ago and dated, as mentioned before.

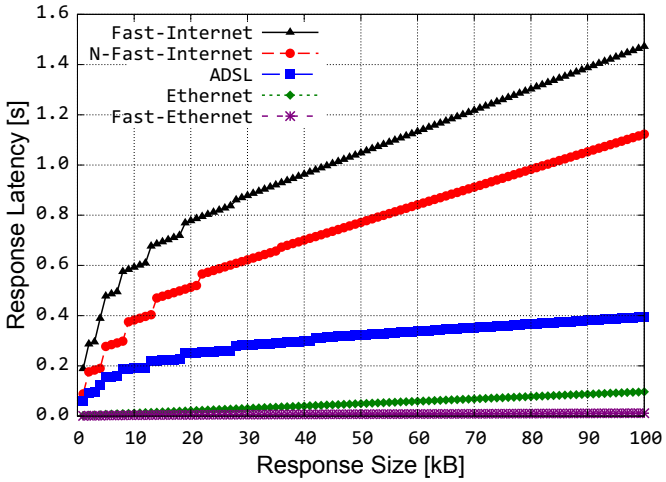

Figure 4.10. Fast network with different response sizes

Furthermore, we compare the performance among persistent connections with or without the pipelining option, representing the HTTP 1.1 version, and the non-persistent connections representing HTTP 1.0. We are able to demonstrate

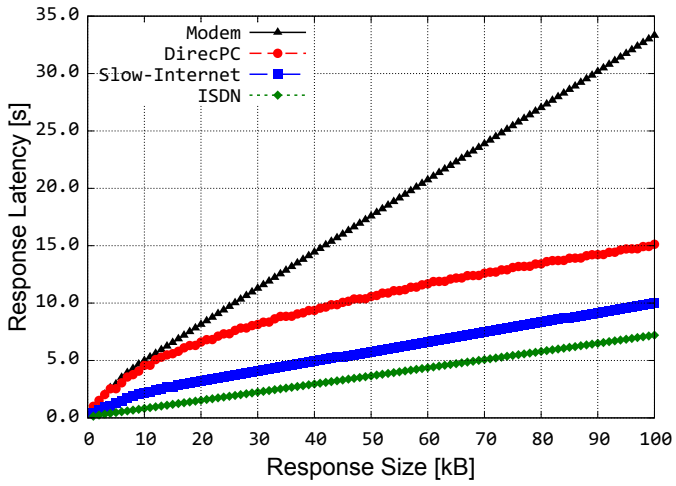

Figure 4.11. Slow network with different response sizes

the performance improvement of HTTP 1.1 with different options. The workload is from 0 to 1000 Web objects with 10 kB in size. The network we use is Fast-Internet. The result is shown in Figure 4.12 with response latency as the metric.

As expected, we can see that HTTP with non-persistent connections performs worst. When the number of Web objects reaches 1000, the response latency is 800 s. The persistent connection without pipelining improves the performance greatly: when carrying same number of Web objects, the latency is only 300 s. Furthermore, when the pipelining option is included in the persistent connection, the latency further drops to 100 s. All the three curves are linear, because all the Web objects have the same size, so as the number of Web objects increases, the latency increases linearly. The slope of the three curves for non-persistent, persistent, and persistent with pipelining are 0.82, 0.28, and 0.09 respectively. The persistent connection option has increased the performance more as it drops the slope from 0.82 to 0.28, while the pipelining option further drops the slope from 0.28 to 0.09. We plan to incorporate the *parallel connection* option in the future performance comparison cases to have a complete list of HTTP options.

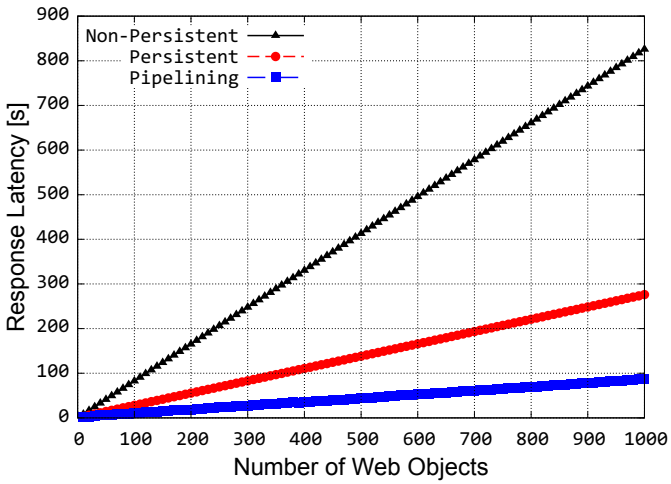

Figure 4.12. Performance of different HTTP versions

We have also tested the different between the wired network and wireless network. As shown in Figure 4.13, the cumulative delay for wireless networks are not significantly highly than that for the wired network, suggesting the usability of HTTP traffic in MANET environment. The difference between the wireless routing protocols are minimal, as they all can deliver the Web objects quite efficiently.

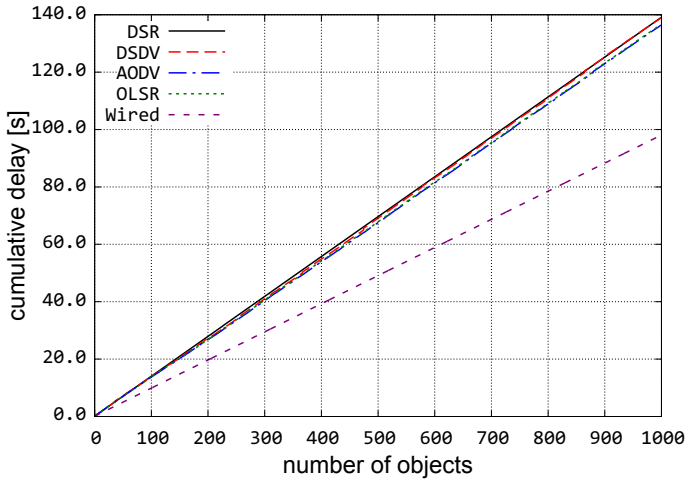

Figure 4.13. Performance of HTTP pipelining in wired and wireless environment

In this chapter we have presented the implementation details of our HTTP traffic generator for the ns-3 network simulator and validated its performance. Our results confirm both the source variable generation functions and latency for different networks when carrying HTTP traffic. We analyse HTTP performance over different network conditions with different response sizes. Our results demonstrate that latency is inversely proportional to response size. The larger the response size is, the larger the latency required to transfer. The performance comparison case among different HTTP versions confirms the improvements of HTTP 1.1 options. As part of future work, we plan to include HTTP with *parallel connection* option.

Chapter 5

Conclusions and Future Work

This chapter provides the concluding remarks in Section 5.1 and highlights the performance comparison results. Section 5.2 considers the future work required to include more protocol comparison work and provides more detailed simulation comparison works.

5.1 Conclusions

This thesis provides a baseline performance comparison work with different combination of application traffic types, transport protocols, routing protocols, and offers guidance to future protocol benchmark and new protocol design.

5.2 Future Work

There is scope for more work to be done on comparing different topological routing protocols and multicast routing protocols. Some other transport protocols can be included for test. For example, different TCP variants, SCPS-TP. Our ResiliNets group is working on SCPS-TP [79] implementation. SCPS-TP is de-

signed for satellite links and is used to deal with high error-rate links. We plan to compare different geographical routing protocol such as SIFT [60] and LAR [59] and epidemic routing [80]. We will also analyse the scalability of MANET by testing large number of nodes and traffic sources in network simulations.

References

[1] M. Conti and S. Giordano. Multihop Ad Hoc Networking: The Theory. *IEEE Communications Magazine*, 45(4):78–86, April 2007.

[2] Josh Broch, David Maltz, David Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. *Proceedings of the 4th annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 85–97, Oct 1998.

[3] S.R. Das, C.E. Perkins, and E.M. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *IEEE INFOCOM*, volume 1, pages 3 –12 vol.1, Tel Aviv, Israel, 2000.

[4] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. MANET simulation studies: the incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(4):50–61, 2005.

[5] C.E. Perkins and E.M. Royer. Ad-hoc On-demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 90–100, February 1999.

[6] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *ACM SIGCOMM*, pages 234–244, London, 1994.

[7] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Tomasz Imielinski and Henry F. Korth, editors, *Mobile Computing*, volume 353 of *The Kluwer International Series in Engineering and Computer Science*, chapter 5, pages 153–181. Kluwer Academic Publishers, Norwood, MA, 1996.

[8] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003.

[9] M.S. Corson V.D. Park. Temporally-Ordered Routing Algorithm (TORA) Version 1: Functional specification. Internet Draft, August 1998.

[10] Jungkeun Yoon, Mingyan Liu, and B. Noble. Random Waypoint Considered Harmful. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1312–1321 vol.2, 2003.

[11] The network simulator: ns-2. `http://www.isi.edu/nsnam/ns/`, December 2007.

[12] The GloMoSim. `http://pcl.cs.ucla.edu/projects/glomosim/`.

[13] The QualNet. `http://www.scalable-networks.com`.

[14] Network simulation (opnet modeler suite). `http://www.riverbed.com/products-solutions/products/network-performance-management/`

network-planning-simulation/Network-Simulation.html, December 1986.

[15] The ns-3 network simulator. http://www.nsnam.org, July 2009.

[16] E. Weingartner, H. vom Lehn, and K. Wehrle. A Performance Comparison of Recent Network Simulators. In *IEEE International Conference on Communications (ICC)*, pages 1–5, June 2009.

[17] Thomas R. Henderson, Sumit Roy, Sally Floyd, and George F. Riley. ns-3 project goals. In *WNS2*, New York, NY, USA, 2006. ACM.

[18] Yunjiao Xue, Ho Sung Lee, Ming Yang, P. Kumarawadu, H.H. Ghenniwa, and Weiming Shen. Performance evaluation of ns-2 simulator for wireless sensor networks. In *CCECE*, pages 1372 –1375, Vancouver, Canada, April 2007.

[19] Hemanth Narra, Yufei Cheng, Egemen K. Çetinkaya, Justin P. Rohrer, and James P.G. Sterbenz. Destination-sequenced distance vector (DSDV) routing protocol implementation in ns-3. In *Proceedings of the ICST SIMUTools Workshop on ns-3 (WNS3)*, pages 439–446, Barcelona, Spain, March 2011.

[20] Dan Broyles, Abdul Jabbar, and James P. G. Sterbenz. Design and analysis of a 3–D gauss-markov mobility model for highly-dynamic airborne networks. In *Proceedings of the International Telemetering Conference (ITC)*, San Diego, CA, October 2010.

[21] Yufei Cheng, Egemen K. Çetinkaya, and James P.G. Sterbenz. Dynamic source routing (DSR) protocol implementation in ns-3. In *Proceedings of*

the ICST SIMUTools Workshop on ns-3 (WNS3), pages 367–374, Sirmione, March 2012.

[22] Yufei Cheng, Egemen K. Çetinkaya, and James P.G. Sterbenz. Transactional Traffic Generator Implementation in ns-3. In *Proceedings of the ICST SIMUTools Workshop on ns-3 (WNS3)*, pages 182–189, Cannes, France, March 2013.

[23] Siddharth Gangadhar, Truc Anh N. Nguyen, Greeshma Umapathi, and James P.G. Sterbenz. TCP Westwood Protocol Implementation in ns-3. In *Proceedings of the ICST SIMUTools Workshop on ns-3 (WNS3)*, Cannes, France, March 2013.

[24] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. *SIGCOMM Comput. Commun. Rev.*, 24(4):24–35, 1994.

[25] S. Mascolo, C. Casetti, M. Gerla, M.Y. Sanadidi, and R. Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 287–297. ACM, 2001.

[26] Thomas D. Dyer and Rajendra V. Boppana. A comparison of tcp performance over three routing protocols for mobile ad hoc networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '01, pages 56–66, New York, NY, USA, 2001. ACM.

[27] T. D Dyer and R. V Boppana. Routing HTTP traffic in a mobile ad hoc network. In *IEEE MILCOM*, volume 2, pages 958–963, Oct. 2002.

[28] T. D Dyer and R. V Boppana. On Routing Web and Multimedia Traffic in Mobile Ad Hoc Networks. In *36th Annual Hawaii International Conference on System Sciences (HICSS)*, Jan. 2003.

[29] Martin F Arlitt and Carey L Williamson. Web Server Workload Characterization: The Search for Invariants. In *ACM SIGMETRICS*, pages 126–137, 1996.

[30] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996.

[31] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[32] F. Donelson Smith, Félix Hernández Campos, Kevin Jeffay, and David Ott. What TCP/IP Protocol Headers Can Tell Us About the Web. In *ACM SIGMETRICS*, pages 245–256, 2001.

[33] Mark E. Crovella and Lester Lipsky. Long-lasting transient conditions in simulations with heavy-tailed workloads. In *Winter Simulation Conferencemulation*, pages 1005–1012, Washington, DC, USA, 1997. IEEE Computer Society.

[34] George S. Fishman and Ivo J. B. F. Adan. How heavy-tailed distributions affect simulation-generated time averages. *ACM Trans. Model. Comput. Simul.*, 16:152–173, April 2006.

[35] M.C. Weigle. Improving Confidence in Network Simulations. In *Winter Simulation Conference*, pages 2188–2194, Dec. 2006.

[36] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On dominant characteristics of residential broadband internet traffic. In *ACM SIGCOMM*, pages 90–102, New York, NY, 2009.

[37] Jeffrey Erman, Alexandre Gerber, Mohammad T. Hajiaghayi, Dan Pei, and Oliver Spatscheck. Network-aware forward caching. In *ACM WWW*, pages 291–300, New York, NY, 2009. ACM.

[38] Jin Cao, W.S. Cleveland, Yuan Gao, K. Jeffay, F.D. Smith, and M. Weigle. Stochastic Models for Generating Synthetic HTTP Source Traffic. In *IEEE INFOCOM*, volume 3, pages 1546–1557, Mar. 2004.

[39] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM*, pages 75–86, New Delhi, India, 2010.

[40] David Kotz and Kobby Essien. Characterizing usage of a campus-wide wireless network. Technical report, ACM MobiCom, Atlanta, Georgia, September 2002.

[41] David Kotz and Kobby Essien. Analysis of a campus-wide wireless network. *Wirel. Netw.*, 11:115–133, January 2005.

[42] John Heidemann, Katia Obraczka, and Joe Touch. Modeling the Performance of HTTP over Several Transport Protocols. *IEEE/ACM Trans. Netw.*, 5(5):616–630, 1997.

[43] Mark E. Crovella and Azer Bestavros. Self-similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Trans. Netw.*, 5:835–846, Dec. 1997.

[44] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the Self-similar Nature of Ethernet Traffic. *IEEE/ACM Networking*, 2(1):1–15, Feb. 1994.

[45] V. Paxson and S. Floyd. Wide Area Traffic: the Failure of Poisson Modeling. *IEEE/ACM Networking*, 3(3):226–244, Jun. 1995.

[46] Hyoung-Kee Choi and J. O Limb. A Behavioral Model of Web Traffic. In *IEEE ICNP*, pages 327–334, Oct. 1999.

[47] Felix Hernandez-Campos, Kevin Jeffay, and F. Donelson Smith. Modeling and Generating TCP Application Workloads. In *BROADNETS*, pages 280–289, Sep. 2007.

[48] Long Le, Jay Aikat, Kevin Jeffay, and F. Donelson Smith. The Effects of Active Queue Management on Web Performance. In *ACM SIGCOMM*, pages 265–276, New York, NY, 2003. ACM.

[49] B. A Mah. An Empirical Model of HTTP Network Traffic. In *IEEE INFOCOM*, volume 2, pages 592–600, Kobe, Japan, 1997.

[50] Paul Barford and Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *ACM SIGMETRICS*, pages 151–160, Madison, Wisconsin, June 1998.

[51] W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson. Self-similarity Through High-variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, February 1997.

[52] Kihong Park. On the Relationship Between File Sizes, Transport Protocols, and Self-Similar Network Traffic. In *IEEE ICNP*, pages 171–180, Columbus, Ohio, Nov. 1996.

[53] Jin Cao, William S. Cleveland, Dong Lin, and Don X. Sun. On the Non-stationarity of Internet Traffic. In *ACM SIGMETRICS*, pages 102–112, New York, NY, Jun. 2001.

[54] E.M. Royer and Chai-Keong Toh. A review of current routing protocols for ad hoc mobile wireless networks. *Personal Communications, IEEE*, 6(2):46–55, 1999.

[55] M. Mauve, A. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network*, 15(6):30–39, 2001.

[56] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003.

[57] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC 4728 (Experimental), February 2007.

[58] Brad Karp and H. T. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 243–254, New York, NY, USA, 2000. ACM.

[59] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. *Wirel. Netw.*, 6(4):307–321, July 2000.

[60] A. Capone, L. Pizziniaco, I. Filippini, and M.A.G. de la Fuente. A SiFT: an Efficient Method for Trajectory Based Forwarding. In *Wireless Communication Systems, 2005. 2nd International Symposium on*, pages 135–139, 2005.

[61] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A distance routing effect algorithm for mobility (DREAM). In *4th annual ACM Mobile computing and networking*, MobiCom '98, pages 76–84, New York, NY, USA, 1998. ACM.

[62] Justin P. Rohrer, Abdul Jabbar, Egemen K. Çetinkaya, Erik Perrins, and James P.G. Sterbenz. Highly-dynamic cross-layered aeronautical network architecture. *IEEE Transactions on Aerospace and Electronic Systems*, 47(4):2742–2765, October 2011.

[63] Justin P. Rohrer, Abdul Jabbar, Egemen K. Çetinkaya, and James P.G. Sterbenz. Airborne telemetry networks: Challenges and solutions in the ANTP suite. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*, pages 74–79, San Jose, CA, November 2010.

[64] Kevin Peters, Abdul Jabbar, Egemen K. Çetinkaya, and James P.G. Sterbenz. A geographical routing protocol for highly-dynamic aeronautical networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, pages 492–497, Cancun, Mexico, March 2011.

[65] Abdul Jabbar and James P. G. Sterbenz. AeroRP: A geolocation assisted aeronautical routing protocol for highly dynamic telemetry environments. In *Proceedings of the International Telemetering Conference (ITC)*, Las Vegas, NV, October 2009.

[66] Kevin Peters, Egemen K. Çetinkaya, and James P. G. Sterbenz. Analysis of a geolocation-assisted routing protocol for airborne telemetry networks. In *Proceedings of the International Telemetering Conference (ITC)*, San Diego, CA, October 2010.

[67] Fan Bai, N. Sadagopan, and A. Helmy. IMPORTANT: a Framework to Systematically Analyze the Impact of Mobility on Performance of Routing Protocols for Adhoc Networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 825–835 vol.2, 2003.

[68] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, 2(5):483–502, 2002.

[69] W. Navidi and T. Camp. Stationary distributions for the random waypoint mobility model. *Mobile Computing, IEEE Transactions on*, 3(1):99–108, 2004.

[70] Sunghwan Ihm and Vivek S. Pai. Towards Understanding Modern Web Traffic. In *ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 295–312, New York, NY, 2011. ACM.

[71] R. Pries, Z. Magyari, and P. Tran-Gia. An HTTP Web Traffic Model based on the Top One Million Visited Web Pages. In *EURO-NGI*, pages 133–139, June 2012.

[72] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[73] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[74] Justin P. Rohrer, Egemen K. Çetinkaya, Hemmanth Narra, Dan Broyles, Kevin Peters, and James P. G. Sterbenz. AeroRP Performance in Highly-Dynamic Airborne Networks using 3D Gauss-Markov Mobility Model. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*, pages 834–841, Baltimore, MD, November 2011.

[75] Yufei Cheng, Egemen K. Çetinkaya, and James P.G. Sterbenz. Performance Comparison of Routing Protocols for Transactional Traffic over Aeronautical Networks. In *Intl. Telemetering Conf. (ITC)*, Oct. 2011.

[76] Kamakshi Sirisha Pathapati, Truc Anh N. Nguyen, Justin P. Rohrer, and James P.G. Sterbenz. Performance Analysis of the AeroTP Transport Protocol for Highly-Dynamic Airborne Telemetry Networks. In *Proceedings of the International Telemetering Conference (ITC)*, Las Vegas, NV, October 2011.

[77] Hemanth Narra, Egemen K. Çetinkaya, and James P.G. Sterbenz. Performance analysis of aerorp with ground station updates in highly-dynamic airborne telemetry networks. In *Proceedings of the International Telemetering Conference (ITC)*, Las Vegas, NV, October 2011.

[78] Henrik Frystyk Nielsen, James Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Håkon Wium Lie, and Chris Lilley. Network Performance Effects of HTTP/1.1, CSS1, and PNG. In *ACM SIGCOMM*, pages 155–166, New York, NY, Oct. 1997.

[79] R.H. Wang and S. Horan. Performance of space communication protocol standards (scps) over acts satellite links. In *IEEE GLOBECOM*, volume 1, page 5 pp., Nov. 2005.

[80] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-200006, Duke University, April 2000.